Bridging the gap between high-level and low-level spatial algorithms

Tien Thao Nguyen

Université Paris-Est
Ecole Doctorale MSTIC

Doctorat en Informatique

Tien Thao NGUYEN

Bridging the Gap from High-level to Low-level Spatial Algorithmics

Sous la direction de Pierre VALARCHER et l'encadrement de Luidnel MAIGNAN

Date de soutenance: 15 décembre 2021

Jury:

Pierre VALARCHER	UPEC	Directeur de thèse
Luidnel MAIGNAN	UPEC	Encadrant de thèse
Jérôme DURAND-LOSE	Université Orléans	Rapporteur
Jean-Louis GIAVITTO	IRCAM	Rapporteur
Véronique TERRIER	GREYC Caen	Examinatrice
Christine EISENBEIS	INRIA Saclay	Examinatrice

Contents

0.1 Résumé en francais

En informatique, il existe une dichotomie entre calculs séquentiels et calculs parallèles, chacun d'eux étant abstrait de diverses manières selon l'objectif. Nous nous intéressons aux calculs parallèles tels que capturés par le modèle des automates cellulaires introduit par John von Neumann dans les années 40. Nous nous concentrons sur les outils théoriques et pratiques nécessaires pour concevoir un automate cellulaire particulier comme solution à un problème algorithmique parallèle donné. Des exemples de tels problèmes sont le problème de synchronisation de fusiliers et les problèmes de génération de séquences en temps réel. Dans la programmation informatique ordinaire, un programmeur réfléchit généralement à un problème, propose un algorithme abstrait et l'implémente dans un langage de programmation de haut niveau qui est transformé par un compilateur en code d'assemblage de bas niveau. Le langage de programmation de haut niveau est censé permettre au programmeur d'exprimer son idée aussi directement ou clairement que possible, et le compilateur optimise généralement le code d'assemblage produit d'une manière ou d'une autre. Considérant les tables de transition d'automates cellulaires comme un code assembleur exécutable de bas niveau, deux approches principales ont été imaginées pour fournir une description formelle de haut niveau de "l'algorithme cellulaire" : les machines à signaux introduites par Jérôme Durand-Lose en 2003, et les champs cellulaires introduit en 2010 par Luidnel Maignan. Une première description grossière du but de cette thèse est de permettre la réduction automatique du nombre d'états et/ou du nombre de transitions d'un automate cellulaire déjà donné, cette réduction automatique étant pensée comme l'étape d'optimisation du "processus de compilation" de l'approche par champs cellulaires en particulier. Ceci est fait en considérant le concept de "simulation locale". Ce dernier permet de transformer un diagramme espace-temps en un autre de manière locale. En transformant des familles de diagrammes espace-temps d'un automate cellulaire, un nouvel automate cellulaire peut ainsi être extrait de la famille de diagrammes résultante. Ce nouvel automate cellulaire est similaire à l'original, seul l'encodage local des informations est modifié. Cela signifie que le nombre d'états ou le nombre de transitions peut changer tout en préservant la correction de l'automate cellulaire par rapport à une spécification spatio-temporelle donnée. Ce concept est appliqué à l'exploration de solutions pour le problème de synchronisation de fusiliers et pour les problèmes de génération de séquences en temps réel. Dans le premier cas, cela montre qu'il existe des millions de solutions à 6 états de la synchronisation, en les générant automatiquement à partir de solutions connues. A noter qu'il s'agit là d'une grande surprise compte tenu de la littérature actuelle sur le sujet. Dans le second cas, cela conduit à la première solution à 4 états pour la génération de la séquence n^3, un résultat qui est obtenu en utiliser aucune compréhension de l'algorithme sous-jacent, mais plutôt en manipulant l'automate cellulaire avec des simulations locales, de façon semi-automatiques.

Mots clés: algorithmique répartie, programmation parallèle, modèle de calculs informatique, automates cellulaires, minimisation d'automates, simulation locale, génération automatique, problème de synchronisation de fusiliers, génération de séquence en temps-réel.

0.2 Abstract

In computer science, there is a dichotomy between sequential computations and parallel computations, each of them being abstracted in various ways depending on the goal. We are interested in parallel computations as captured by the model of cellular automata introduced by John von Neumann in the 40's. We focus on theoretical and practical tools needed to design a particular cellular automaton as a solution for a given parallel algorithmic problem. Examples of such problems are the firing squad synchronization problem and the real-time sequence generation problems. In ordinary computer programming, a programmer typical thinks about a problem, comes up with an abstract algorithm, and implement it in a high-level programming language that is transformed by a compiler to the low-level assembly code. The high-level programming language is supposed to allow the programmer to express his or her idea as directly or clearly as possible, and the compiler typically optimizes the produced assembly code in some way. Thinking of transition tables of cellular automata as a low-level executable assembly code, two main approaches have been devised to provide a high-level formal description of "cellular algorithm": signal machines introduced by Jérôme Durand-Lose in 2003, and cellular fields introduced in 2010 by Luidnel Maignan. A first rough description of the goal of this thesis is to allow automatic reduction of the number of states and/or the number of transitions of an already given cellular automaton, this automatic reduction being thought of as the optimization step of the "compilation process" of cellular field approach in particular. This is done by considering the concept of "local simulation". The latter allows to transform a space-time diagram into another one in a local manner. By transforming families of space-time diagrams of a cellular automaton, a new cellular automaton can thus be extracted from the resulting family of diagrams. This new cellular automaton is similar to the original one, only the local encoding of information is changed. This means that the number of states or number of transitions can change while preserving the correctness of the cellular automaton with respect to a given space-time specification. This concept is applied to the exploration of solutions for the firing squad synchronization problem and for the real-time sequence generation problems. In the first case, this shows that there are millions of 6-state synchronization solutions, by generating them automatically from known solutions. Note that this is a big surprise given the current literature on the subject. In the second case, it leads to the first 4-state solution for the generation of the n^3 sequence, a result which is obtained with no use of any understanding of the algorithm, but instead by manipulating the cellular automaton with local simulations, in a semi-automatic manner.

Keywords: distributed computing, parallel computing, models of computation, cellular automata, automata minimization, local simulation, automatic generation, firing squad synchronization problem, real-time sequence generation.

Chapter 1

Cellular Automata Algorithms

In computer science, there is a dichotomy between sequential computations and parallel computations, each of them being abstracted in various ways depending on the goal. We are interested in parallel computations as captured by the model of Cellular Automata (CA for short) introduced by John von Neumann in the 40's [37] (Section 1.1). We focus on theoretical and practical tools needed to design a particular CA as a solution for a given parallel algorithmic problem. Examples of such problems are the firing squad synchronization problem and the real-time sequence generation problems (Section 1.2).

In ordinary computer programming, a programmer typical thinks about a problem, comes up with an abstract algorithm, and implement it in a high-level programming language that is transformed by a compiler to the low-level assembly code. The high-level programming language is supposed to allow the programmer to express his or her idea as directly or clearly as possible, and the compiler typically optimizes the produced assembly code in some way. Thinking of CA transition tables as a low-level executable assembly code, two main approaches have been devised to provide a high-level formal description of "cellular algorithms": signal machines introduced by Jérôme Durand-Lose in 2003 [5], and cellular fields introduced during the PhD thesis of Luidnel Maignan [14] (Section 1.3).

A first rough description of the goal of this thesis is to allow automatic reduction of the number of states and/or the number of transitions of an already given cellular automaton, this automatic reduction being thought of as the optimization step of the "compilation process" of cellular field approach in particular.

1.1 Cellular Automata, Tightly

Intuitively speaking, a cellular automaton is a set of rules describing the local, synchronous and uniform evolution of an array of identical cells having a finite number of possible states. Each rule specifies the next state of a cell, given its current state and those of its neighbors. Here is a representation of the 8 rules of the so-called "traffic" cellular automaton.

Figure 1.1: Local transition function δ_τ of the "traffic" cellular automaton τ.

For a given configuration of the array of cells, *i.e.* the choice of a state for each cell, the next configuration is obtained be assigning to each cell the state dictated by the set of rules based on its current state and those of its immediate neighbors. If we represent an initial configuration horizontally, and the resulting configurations on the next line iteratively, we obtain the so-called space-time diagram of the cellular automaton on the given initial configuration. Here is a space-time diagram of the traffic cellular automaton on an initial configuration having 12 cells in the black states. Note how the 8 rules are used to compute this space-time diagram from the chosen initial configuration.

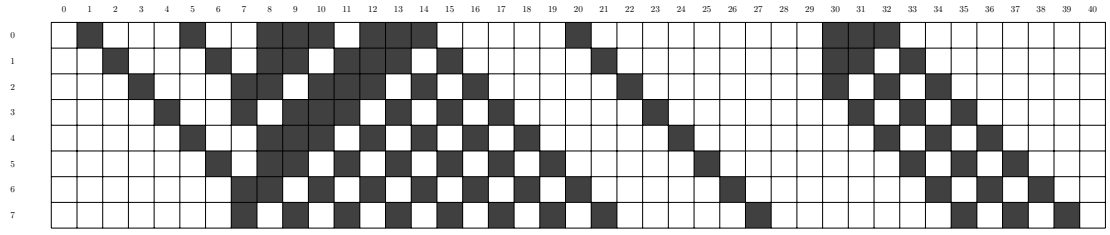


Figure 1.2: Space-time diagram of "traffic" CA τ on a given initial configuration.

A detailed introduction to CA is beyond the scope of this document and we assume the reader to be familiar with the topic. But CA are used in various slightly different ways in different work, so let us give immediately a precise formal definition that fits the need of this document, and better links with the choices in the associated ublications. In particular, we consider uni-dimensional CA with a neighborhood of radius 1. Further generalizations are easy to make once this case well understood. We are also interested in counting the number of useful rules so we have the following definition.

Definition 1. *A* cellular automaton α *consists of a finite set of* states Σ_α*, a set of* initial configurations $\mathrm{I}_\alpha \subseteq {\Sigma_\alpha}^{\mathbb{Z}}$ *and a partial function* $\delta_\alpha : {\Sigma_\alpha}^{\{-1,0,1\}} \nrightarrow \Sigma_\alpha$ *called the* local transition function *or* local transition table *such that for any initial configuration* $c \in \mathrm{I}_\alpha$ *and any* $t \in \mathbb{N}$, $(\delta_\alpha^{\mathbb{Z}})^t(c)$ *is defined. The latter notation* $\delta_\alpha^{\mathbb{Z}}$ *refers to the partial function* $\delta_\alpha^{\mathbb{Z}} : {\Sigma_\alpha}^{\mathbb{Z}} \nrightarrow {\Sigma_\alpha}^{\mathbb{Z}}$ *called the* global transition function *of* α *and defined as*

$$\delta_\alpha^{\mathbb{Z}}(c)(p) = \delta_\alpha(c(\!|p|\!)) \qquad \text{for any } (c,p) \in \mathrm{I}_\alpha \times \mathbb{Z}$$

where $-(\!|-|\!) : \mathrm{I}_\alpha \times \mathbb{Z} \to {\Sigma_\alpha}^{\{-1,0,1\}}$ *is defined as*

$$c(\!|p|\!)(i) = c(p+i) \qquad \text{for any } (c,p,i) \in \mathrm{I}_\alpha \times \mathbb{Z} \times \{-1,0,1\}.$$

The elements of ${\Sigma_\alpha}^{\mathbb{Z}}$ *are called* global configurations*, or just* configurations *for short, and those of* ${\Sigma_\alpha}^{\{-1,0,1\}}$ *are called* local configurations*.*

Let us insist on the fact that the transition functions δ_α and $\delta_\alpha^{\mathbb{Z}}$ might be partial but the global configurations (elements of ${\Sigma_\alpha}^{\mathbb{Z}}$) are total functions on the domain $\mathbb{Z}$ and local configurations (elements of ${\Sigma_\alpha}^{\{-1,0,1\}}$) are total functions on the domain $\{-1,0,1\}$. Given a local configuration $l \in {\Sigma_\alpha}^{\{-1,0,1\}}$, the values $l(-1)$, $l(0)$ and $l(1)$ respectively refer to state of the left, center and right cell of the local configuration. The constraint that we put on the (partial) local transition function δ_α means that the global transition function $\delta_\alpha^{\mathbb{Z}}$ should be applicable any number of times t on any initial configuration c and still give a total configuration $(\delta_\alpha^{\mathbb{Z}})^t(c) \in {\Sigma_\alpha}^{\mathbb{Z}}$. We

mostly restrict our attention to those cellular automata where every transition rule is useful, i.e. $\forall\, l \in \mathrm{dom}(\delta_\alpha), \exists (c,t,p) \in \mathrm{I}_\alpha \times \mathbb{N} \times \mathbb{Z}$, $(\delta_\alpha^{\mathbb{Z}})^t(c)(\!|p|\!) = l$.

Definition 2. *Given a cellular automaton α and an initial configuration $c \in \mathrm{I}_\alpha$, the* space-time diagram $\mathrm{D}_\alpha(c) : \mathbb{N} \times \mathbb{Z} \to \Sigma_\alpha$ *is defined as:*

$$\mathrm{D}_\alpha(c)(t,p) = (\delta_\alpha^{\mathbb{Z}})^t(c)(p) \qquad \text{for any } (t,p) \in \mathbb{N} \times \mathbb{Z}.$$

When $\mathrm{D}_\alpha(c)(t,p) = s$, we say that, for the cellular automaton α and initial configuration c, the cell at position p has state s at time t.

In the above example of the traffic cellular automaton τ, the set of states Σ_τ is $\{0,\ 1\}$ (0 for white empty cells, and 1 for filled black cells), and all configurations are possible initial configurations, *i.e.* $\mathrm{I}_\tau = {\Sigma_\tau}^{\mathbb{Z}}$. The local transition function δ_τ is therefore a total function, and so is the global transition function $\delta_\tau^{\mathbb{Z}}$. Note that cells at position less than 0 or bigger than 40, those not represented in the space-time diagrams, are all in the white state 0. But our definition of cellular automata allows to restrict the set of initial configurations to, say, configurations where there is never more than two consecutive black cell. This leads to another cellular automaton τ' with the same set of states $\Sigma_{\tau'} = \{0,1\}$ and with $\mathrm{I}_{\tau'} = \{c \in \{0,1\}^{\mathbb{Z}} \mid \forall\, p \in \mathbb{Z}, (c(p), c(p+1), c(p+2)) \neq (1,1,1)\}$. In this case, the local transition function $\delta_{\tau'}$ can stay undefined on triplet of three black cells $(1,1,1)$.

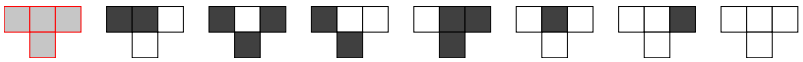


Figure 1.3: Local transition (partial) function $\delta_{\tau'}$ of the CA τ'.

Indeed, such a local configuration never occur in any space-time diagrams from any initial configuration in $\mathrm{I}_{\tau'}$, as in the following example. Put differently, restricting $\delta_{\tau'}$ not to be defined on $(1,1,1)$ implies that $\mathrm{I}_{\tau'}$ also have to restricted at least as we just did in order for the property of having only total configurations to hold.

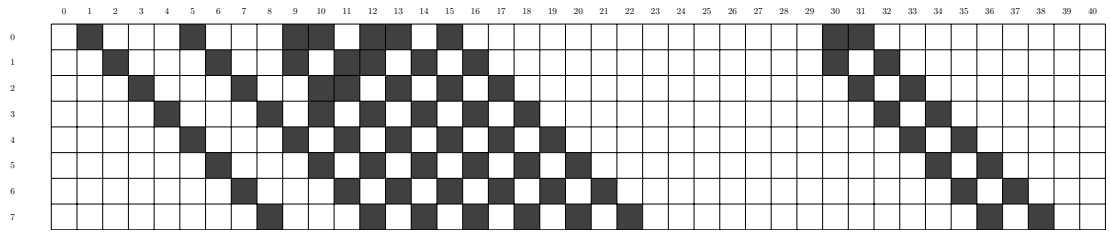


Figure 1.4: Space-time diagram of CA τ' on a given initial configuration in $\mathrm{I}_{\tau'}$.

We can restrict further the set of initial configurations to those where there are no consecutive black cells. This leads to yet another CA τ'' with the same set of states $\Sigma_{\tau''} = \{0,1\}$ and with $\mathrm{I}_{\tau''} = \{c \in \{0,1\}^{\mathbb{Z}} \mid \forall\, p \in \mathbb{Z}, (c(p), c(p+1)) \neq (1,1)\}$. In this case, the local transition function $\delta_{\tau''}$, depicted in Fig 1.5, can stay undefined on even more local configurations. Only 5 rules suffice has exemplified in Fig 1.6.

These examples are a bit misleading as they have the special property that any configuration occurring in the space-time diagrams also belong to the set of initial configurations. This is not required as will become clear thereafter. As a last note, we will consider the following vocabulary in relation with cellular fields later on.

Figure 1.5: Local transition (partial) function $\delta_{\tau''}$ of the CA τ''.

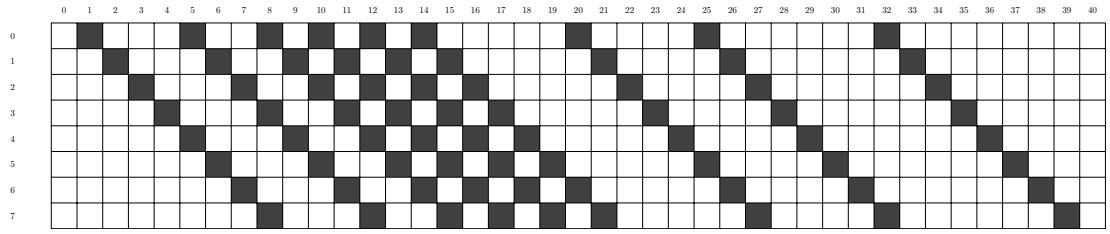


Figure 1.6: Space-time diagram of CA τ'' on a given initial configuration in $\mathrm{I}_{\tau''}$.

Definition 3. *An* infinite cellular automaton *is almost a cellular automaton, except that the set of states is not necessarily finite.*

The benefit of these definitions is that we can associate to any such (finite) CA a number of states and a number of transitions. We are interested in procedures allowing, for instance, to start with an infinite CA, to transform it into a finite CA with a given number of states and transitions, then to transform this finite CA into another one with fewer states or fewer transitions, and so on so forth. But these transformations have to preserve the important properties of the CA, namely their correctness as solutions to a given algorithmic problem, in the same way that a compiler might optimize the size of an assembly code or its memory usage while preserving its correctness.

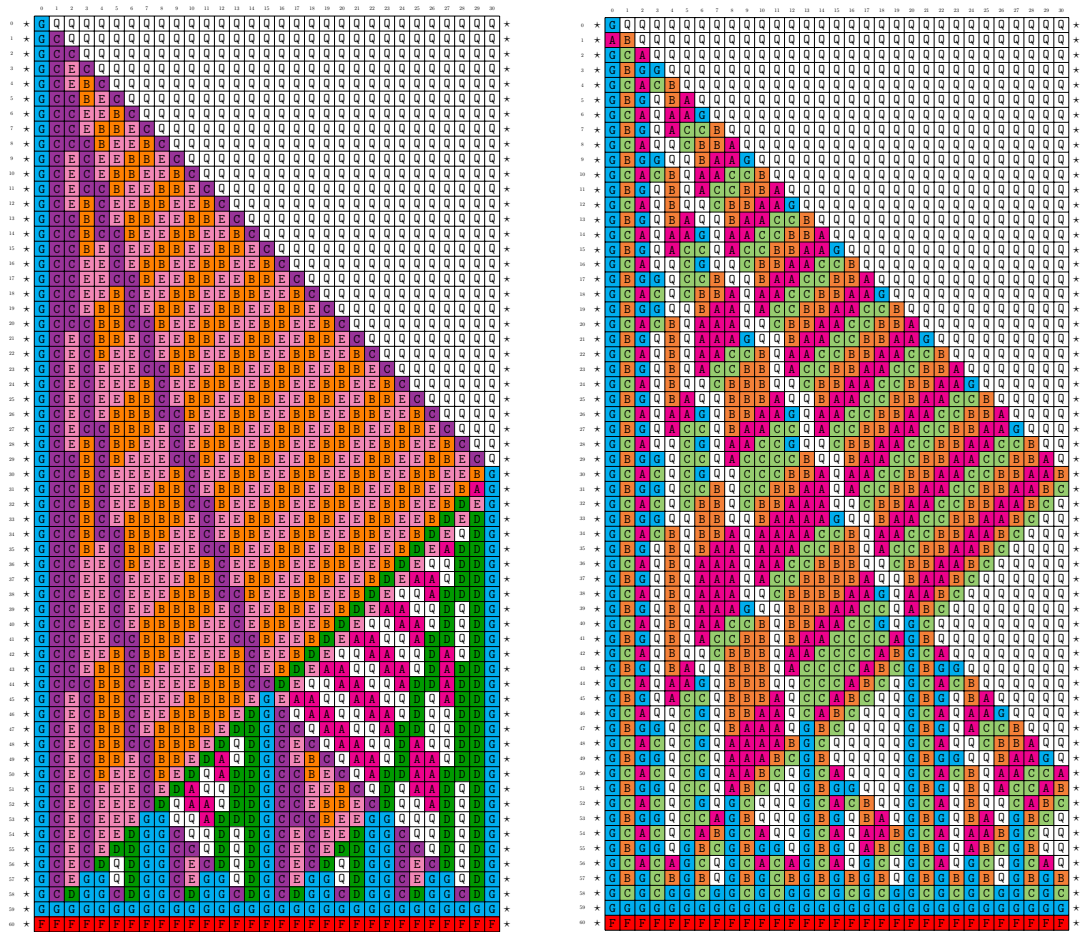
1.2 Algorithmic Problems

Since von Neumann's studies on auto-replication and synchronization of CA [37], a number of algorithmic problems have been considered. In this work, we use two algorithmic problems as examples: the firing squad synchronization problem, and the problem of real-time sequence generation.

1.2.1 The Firing Squad Synchronization Problem

The Firing Squad Synchronization Problem (FSSP for short) was proposed by John Myhill in 1957 [23]. The goal is to find a single cellular automaton that synchronizes any one-dimensional horizontal array of an arbitrary number of cells. More precisely, one considers that at initial time, all cells are inactive (i.e. in a special *quiescent* state) except for the leftmost cell which is active (i.e. in a special *general* state). One wants the local transition function of the cellular automaton to lead all cells to transition to a special *synchronization* or *firing* state **for** the first time **at** the same time. This time t_s is called the *synchronization time* and it is known that its minimal possible value is $2n - 2$ where n is the number of cells [23].

For this problem, many minimal-time solutions were proposed using different approaches. As indicated in [32], the first one was proposed by Goto in 1962 [32] with



(a) Balzer 8-state 165-rule [1] (b) Mazoyer 6-state 119-rule [21]

Figure 1.7: Space-time diagram of Balzer's and Mazoyer's minimal-time solutions.

many thousands of states, followed by Waksman in 1966 [38], Balzer in 1967 [1], Gerken in 1987 [7], and finally Mazoyer in 1987 [21] who presented respectively a 16-state, 8-state, 7-state and 6-state minimum-time solution, with no further improvements in number of states since 1987. Balzer [1] shows that there are no 4-state minimal-time solutions, latter confirmed by Sanders [30] through an exhaustive search and some corrections to Balzer's work. Whether there exists any 5-state minimum-time solution or not is still an open question. In Fig 1.7, one can see the two archetypal strategies to obtain a minimum-time FSSP solution.

In fact, all these solutions use a "divide and conquer" strategy. Goto's solution were pretty complex with two types of divisions. The following ones used a "mid-way" division but Mazoyer's 6-state solution uses for the first time a "two-third" type of division[1]. Until recently, it was believed that the special "two-third" type of division used by Mazoyer was necessary to achieve as few as 6 states. In other words, it was believed that one needed to change the "'mid-way" algorithm to improve the number of states. But in 2018, Clergue, Verel and Formenti [3] generated 718 new 6-state solutions using an Iterated Local Search algorithm to explore the space of 6-state solutions on a cluster of heterogeneous machines: 717 of these solutions use a "mid-way" division, and only one use a "two-third" division. This came as

[1]See Figure 1.7a for a mid-way division, and Figure 1.7b for a two-third division

a surprise, firstly because "mid-way" division seems now to be the norm even for 6-state CA, but also because 718 solutions seemed to be a lot of solutions. In this document, we show how it is possible to find millions of such solutions with a single personal computer [26] ! This was discovered in the search for an semi-automatic way to go from an infinite CA to a 6-state CA [25], but this asks also the question of whether a similar technique can be used to find a 5-state CA in order to close the aforementioned long standing open question of existence of such solutions. Some hope in this direction comes from our result for the following algorithmic problem [28], but we are getting ahead of ourselves.

Of course, there are also non-minimal-time solutions of interest [35]. In 1965, Fischer proposed an algorithm synchronizing at time $3n-4$ [6] et Umeo proposed an implementation using 15 states and 188 rules [35]. 1967, Minsky et McCarthy [22] proposed an $3n + O(\log n)$ algorithm and Yunès proposed an implementation using 13 states and 138 rules in 1994 [39], together with two 7 states solutions using 134 rules. Other solutions with the same time complexity were proposed by Herman in 1972 [8] (10 states, 155 rules), by Umeo, Maeda et Hongyo in 2006 [34] (6 states, 78 rules), by Yunès in 2008 (6 states, 125 rules), and by Umeo in 2015 [35] (2 solutions of 6 states, one with 114 rules and the other 100 rules). These solutions also differ by other criteria such as their state-change complexity. Also in 2007, Umeo et Yanagihara [36] proposed a partial solution for $n = 2^k$ using 5 states and 67 rules to synchronize at time $3n-3$. Despite the fact that they are not minimal-time, these solutions are also based on a very similar "divide and conquer" strategy. Figure 1.8 illustrates how the non-minimal-time algorithm works. Figure 1.8a shows the space-time diagram of Yunès's solution 7 state, 134 rules and Figure 1.8b shows which of Umeo's solution 6 states, 78 rules.

Let us give a formal definition fitting our framework. In particular, the finite configurations are embedded into infinite ones using an additional special state.

Definition 4. *A cellular automaton α is an* FSSP solution *if there are four special states $\star_\alpha, \mathtt{G}_\alpha, \mathtt{Q}_\alpha, \mathtt{F}_\alpha \in \Sigma_\alpha$ satisfying the following conditions:*

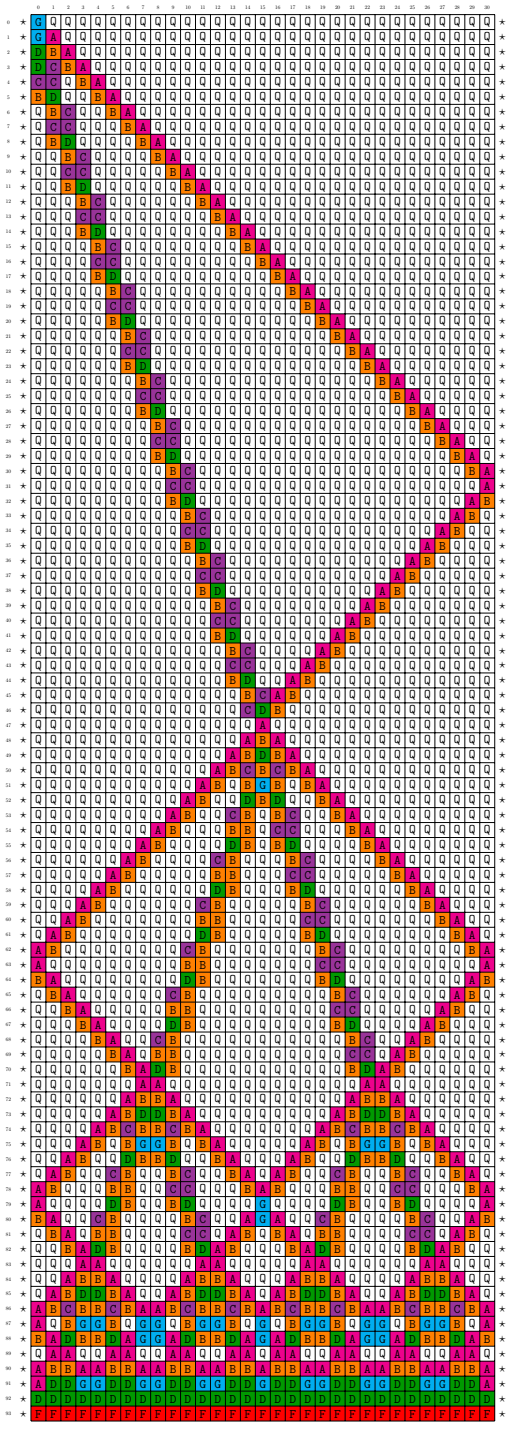
1. *$\star_\alpha$ is an* outside state*,* i.e. $\forall\, l \in dom(\delta_\alpha)$, $\delta_\alpha(l) = \star_\alpha \Leftrightarrow l(0) = \star_\alpha$.

2. *$\mathtt{G}_\alpha$ is a* general state*,* i.e. $\mathrm{I}_\alpha = \{\overline{n}_\alpha \mid n \geq 2\}$ *with $\overline{n}_\alpha \in {\Sigma_\alpha}^{\mathbb{Z}}$ being the* FSSP initial configuration of size n *defined as $\overline{n}_\alpha(p) = \star_\alpha$, $\mathtt{G}_\alpha$, $\mathtt{Q}_\alpha$, $\star_\alpha$ for $p < 0$, $p = 0$, $0 < p < n$, and $p \geq n$ respectively.*

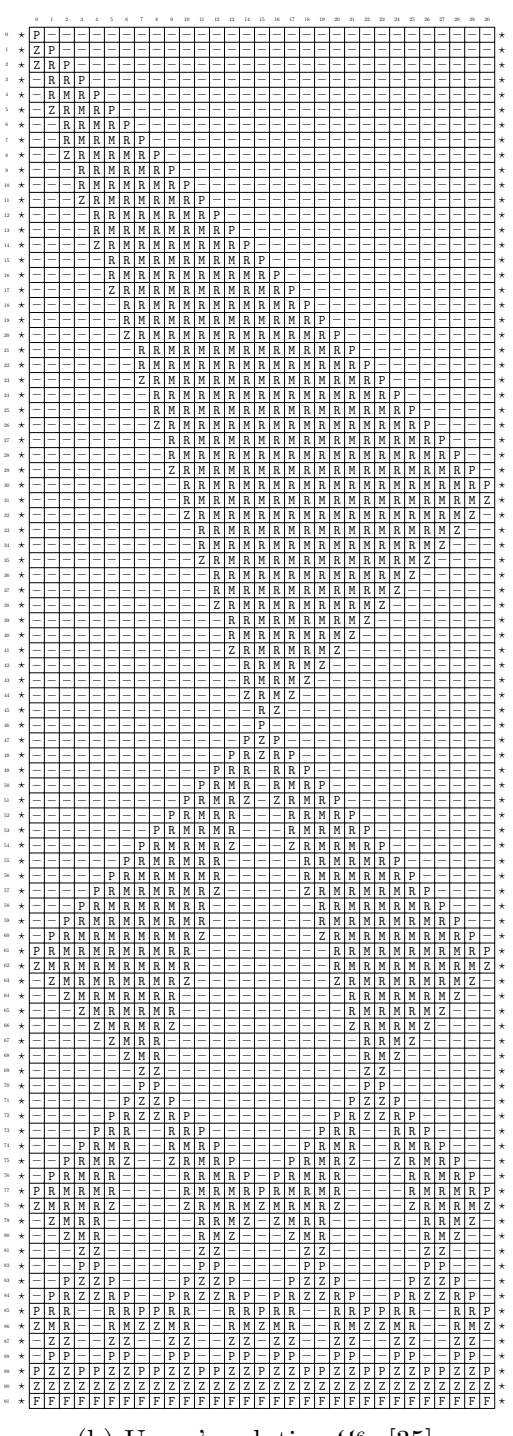
3. *$\mathtt{Q}_\alpha$ is a* quiescent state*,* i.e. $\delta_\alpha(\mathtt{Q}_\alpha, \mathtt{Q}_\alpha, \mathtt{Q}_\alpha) = \delta_\alpha(\mathtt{Q}_\alpha, \mathtt{Q}_\alpha, \star_\alpha) = \mathtt{Q}_\alpha$.

4. *$\mathtt{F}_\alpha$ is a* firing state*,* i.e. $\forall\, n, \exists\, t \in \mathbb{N}$, $\forall (t', p) \in \mathbb{N} \times \mathbb{Z}, \mathrm{D}_\alpha(\overline{n}_\alpha)(t', p) = \mathtt{F}_\alpha \Leftrightarrow t' \geq t$.

The $\star_\alpha$ state is not really counted as a state since it represents cells that should be considered as non-existing. Therefore, an FSSP solution α is said to have s states when $\mid \Sigma_\alpha \setminus \{\star_\alpha\} \mid = s$, and m transitions when $\mid \mathrm{dom}(\delta_\alpha) \setminus \Sigma_\alpha \times \{\star_\alpha, \mathtt{F}_\alpha\} \times \Sigma_\alpha \mid = m$. In the last expression, note that we also uncount the transition happening after the synchronization, *i.e.* those from the firing state to the firing state imposed by the forth condition of FSSP solutions.



(a) Yunes 7-state 134-rule [35]



(b) Umeo's solution $\mathfrak{U}_{78}^{6}$ [35]

Figure 1.8: Space-time diagram of Yunes's and Umeo's non-minimal-time solutions.

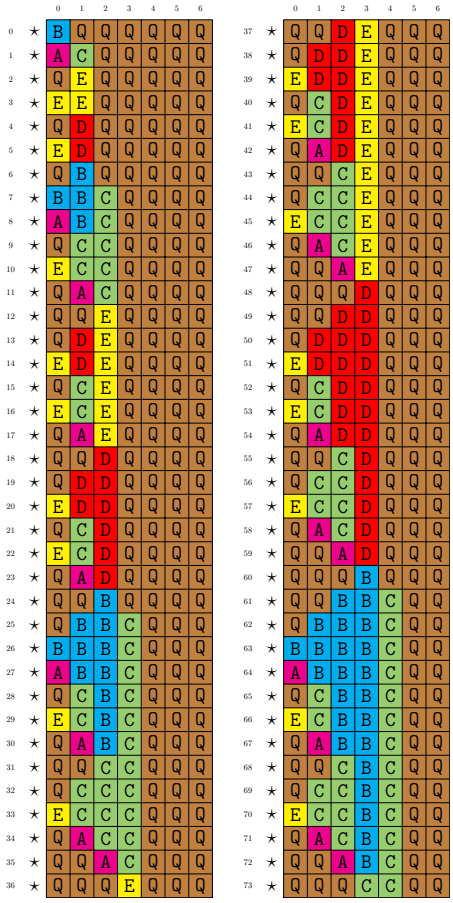


Figure 1.9: Space-time diagram of Kamikawa and Umeo's n^3-RTSG solution.

1.2.2 Real-Time Sequence Generators

Let us consider now another algorithmic problem, the so-called Real-Time Sequence Generation problems (RTSG for short). In the latter, given a fixed sequence $S \subseteq \mathbb{N}$, the goal is to find a cellular automaton running on an one-dimensional horizontal array of cells such that the leftmost cell is in a special state exactly when the number of transition t from the beginning belongs to S. In the following, we write $f(n)$ to mean $S = \{f(n) \mid n \geq 1\}$.

The study of such problems began in 1965 for the sequence of prime numbers, with a description of a cellular automaton algorithm by Fischer [6]. In 1998, Korec [13] proposed a 9-state solution. Other sequences where considered in 2007 by Kamikawa and Umeo [24] who gave some different algorithms for the sequences 2^n, n^2, and 3^n using one-bit inter-cell-communication cellular automata. In 2012, Kamikawa and Umeo [10] described the sequence generation powers of CAs having a small number of states, focusing on the CAs with one (only one sequence n of all positive natural numbers), two, and three internal states, respectively. The authors enumerate all of the sequences generated by two-state CAs (linear sequences: $2n, 4n, 3n-1, n, 3n-2, 2n-1, n+1$; non-regular sequences: $2^{n+1}-2, 2^n-1$) and present several non-regular sequences like $2^n, n^2, 3^n$ that can be generated in real-time by three-state CAs, but not generated by any two-states CA. In 2016 [11], they gave a construction for the Fibonacci sequence using five-state, followed in 2019 [12] by two solutions of 8 states and 6 states for the sequence n^3. Figure 1.9 shows the

solution 6 states of Umeo for the sequence n^3. In these studies, much attention has been paid to the developments of real-time generation algorithms and their small-state implementations on CAs for specific non-regular sequences. Other complexities are also studied such as the space, communication or state-change complexities.

The interesting thing about the latter sequence n^3 is its similarity with the FSSP current situation: the best known solution uses 6 states, and the question of existence of a better solution was open, but the methods describe in this document close the question with an semi-automatic optimization of Umeo's 6-state solution into a 4-state solution ! This shows that the method might have some chance to be useful for the harder open problem of the existence of 5-state FSSP solutions.

Let us give a formal definition for this problem too.

Definition 5. *Given a sequence $S \subseteq \mathbb{N}$, a cellular automaton is a S-*RTSG solution *if there are four special states $\star_\alpha, \mathtt{B}_\alpha, \mathtt{Q}_\alpha, \mathtt{S}_\alpha \in \Sigma_\alpha$ satisfying the following conditions.*

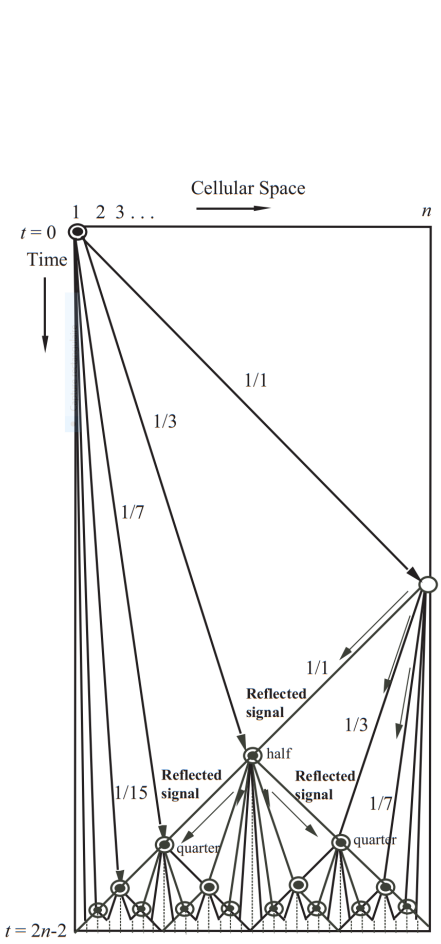
1. *$\star_\alpha$ is an* outside state*, i.e. $\forall\, l \in dom(\delta_\alpha)$, $\delta_\alpha(l) = \star_\alpha \Leftrightarrow l(0) = \star_\alpha$.*
2. *$\mathtt{B}_\alpha$ is the* lauching state*, i.e. $\mathrm{I}_\alpha = \{\overrightarrow{\infty}_\alpha\}$ with $\overrightarrow{\infty}_\alpha$ being the* RTSG initial configuration *defined as $\overrightarrow{\infty}_\alpha(p) = \star_\alpha$, $\mathtt{B}_\alpha$ and $\mathtt{Q}_\alpha$ for $p < 0$, $p = 0$ and $p \geq 1$ respectively.*
3. *$\mathtt{Q}_\alpha$ is a* quiescent state*, i.e. $\delta_\alpha(\mathtt{Q}_\alpha, \mathtt{Q}_\alpha, \mathtt{Q}_\alpha) = \mathtt{Q}_\alpha$ on one hand, and $\delta_\alpha(\star_\alpha, \mathtt{Q}_\alpha, \mathtt{Q}_\alpha) = \mathtt{Q}_\alpha$ if it is defined, on the the hand.*
4. *$\mathtt{S}_\alpha$ is a* generating state*, i.e. $\forall\, t \in \mathbb{N}$, $\mathrm{D}_\alpha(\overrightarrow{\infty}_\alpha)(t, 0) = \mathtt{S}_\alpha \Leftrightarrow t \in S$.*

As for FSSP, the $\star_\alpha$ state is not really counted as a state since it represents cells that should be considered as non-existing. Therefore, an RTSG solution α is said to have s states when $\mid \Sigma_\alpha \setminus \{\star_\alpha\} \mid = s$, and m transitions when $\mid \mathrm{dom}(\delta_\alpha) \setminus \Sigma_\alpha \times \{\star_\alpha\} \times \Sigma_\alpha \mid = m$.

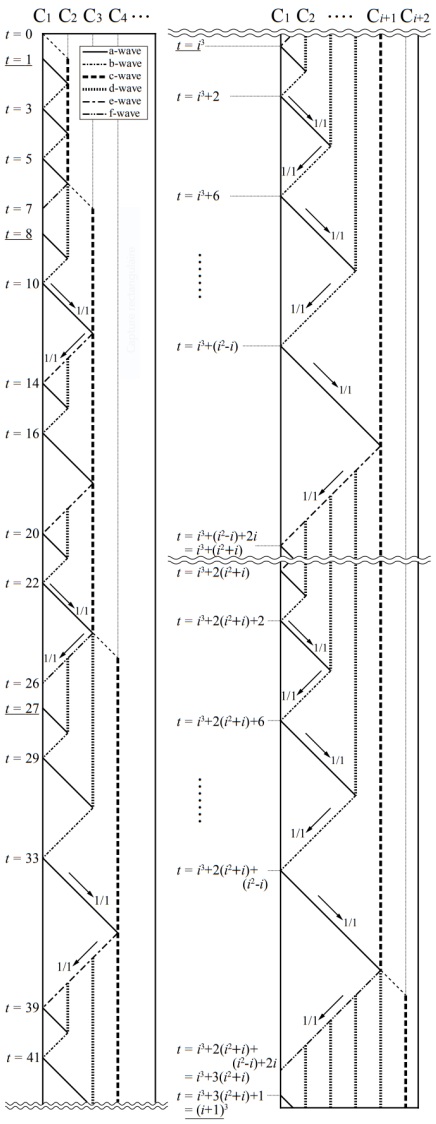
To finish this section, let us summarize. At this point, we almost have all the pieces together to have a full analogy between CA framework and usual computer programming languages and their compilers. We consider CA as low-level assembly code, and the optimization process of CA have to preserve correctness with respect to the considered algorithmic problem as compilers do. With this section, we have just presented the two algorithmic problems that we use as examples. The next section introduce the last piece which should be compared to high-level programming languages used to produce an initial low-level code via the compiler.

1.3 High-Level Descriptions of CA

It is often the case that solutions to CA algorithmic problems are first solved abstractly through some idealized thinking in a first step, this abstract thinking being translated into a concrete CA implementation "by hand" in a second step. It is very tempting to try to formalize the abstract thinking into a kind of high-level framework and to generate automatically the implementation from it, similar to high-level languages compilers. One benefit is that the correctness could in principle be proved on the high level description, making it closer to the designers thinking, and therefore easier to follow. The generation process should then automatically preserve correctness.



(a) FSSP algorithm of [31]



(b) RTSG algorithm of [12]

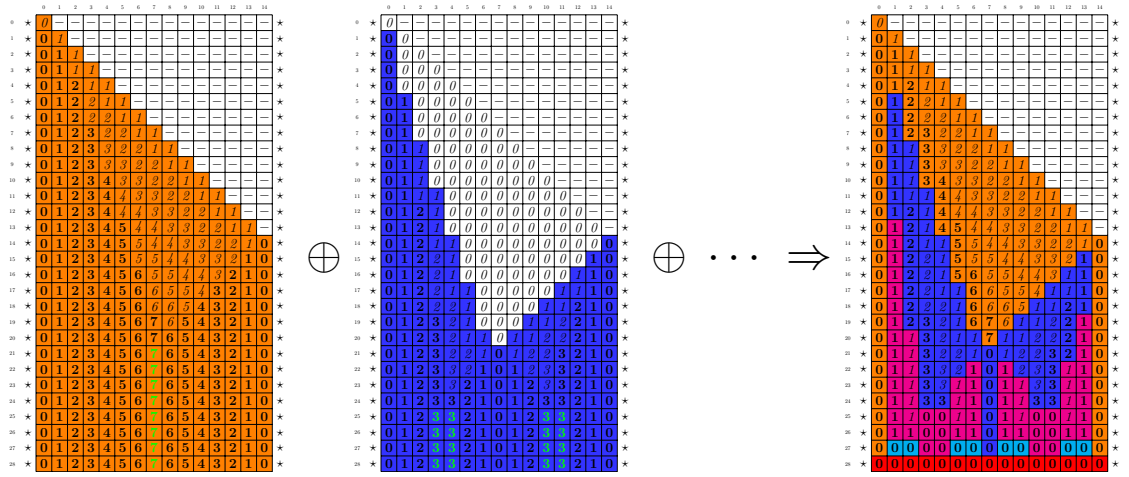
Figure 1.10: Cellular algorithms through signals in a continuous space

1.3.1 Abstract Geometrical Computations

It is a fact that many CA solutions are designed by first thinking in terms of so-called "signals": (finite) information that are sent at a given speed through the cellular space. For simplicity, the discrete nature of the space is often ignored initially, and replaced by a continuous space either. It is in particular true in the case of FSSP and RTSG solutions, for which algorithms are often described geometrically, as shown in the examples of Figure 1.10.

The goal of signal machines is to formalize this common geometrical thinking. The initial configurations of cellular automata are replaced by a set of position/signal pairs and the transition rules are replaced by a speed for each signal and a set of rules specifying which signals should come out when a given set of signals meet on the same exact position. The space-time diagram of such systems are therefore continuous in space and in time as expected.

Although cellular automata geometry is discrete, signal machines idealize this



(a) Level 0 (b) Level 1 (c) All levels

Figure 1.11: Space-time diagrams of two instances of the compound "splitting" field and the (partially reduced) resulting infinite cellular automaton. The first instance is a cellular automaton, *i.e.* receives no external information, but each following instances receives information from the previous one.

geometry into a continuous one to allow a direct expression of the geometrical ideas and when possible, discretization of the "signals" are automatically dealt with as a last step to obtain a transition table as described in [2]. A description of abstract geometrical computations is beyond the scope of this document. The historical motivation of this work is mainly the following second approach. Nonetheless, the main results of this work is not tied to any particular approach.

1.3.2 Decomposition into Cellular Fields

In cellular fields, the space is still discrete but the number of states is typically allowed to be infinite, in order to include more structure and more semantic. Indeed, the main focus is on modularity: a "field" is almost like a cellular automaton, but it can communicate with other "fields". In the case of the FSSP, one can consider a "distance field", *i.e.* a "cellular automaton" asking at each cell if the cell is a delimiting border and computing the distance to those borders. On top of that, one can build the "middle field", *i.e.* a "cellular automaton" asking at each cell the distance to the border and determining the cells that are the most far away from the delimiting borders. Combining these two fields and a few others, one obtains a compound field that starts with a number of delimited regions, and produces twice as much delimited regions. Composing recursively infinitely many instances of this compound field, one obtain a infinite cellular automaton solving the FSSP as pictured in Figure 1.11. The reduction to a finite (and possibly small) number of state is left as a last step in the design.

This approach was initiated by Gruau and Maignan [14] and applied to FSSP in the work of Maignan and Yunès [15, 18]. In the latter, a complete formal description from fields to a finite cellular automaton is described. However, the transition tables thus obtained uses 21 states, a large number of states compared to the 6 states obtained "by hand". But the goal was just to obtain a finite cellular automaton, the hope being to use automatic techniques to optimize the number of states up to

competitive number.

To summarize, the infinite solution being a modular one, a semantical high-level formal description was achieved in these previous works. One benefit is that the same modules arranged differently was expected to describe other algorithms, as in classical computer programming. Other benefits are to have semantical proof of correctness together with a correctness-preserving reduction procedure into a finite state CA using a particular kind of cellular field, "reductions". In other words, the proofs are free of all the intricacies of the "optimization to a finite number of states". This can be applied to ease the formal proof of correctness of Mazoyer's solution for instance. Up to our knowledge, it is known to be long and hard but also to be the only proof to be precise enough to actually be implemented in the Coq Proof Assistant [4].

1.4 Contributions of this Thesis

From theses works and concepts, two intertwined research directions emerge. One direction is to ask whether a reduction to fewer states is firstly possible, and secondly automatically generable, in the same spirit as compiler optimization, with the possible application of reducing further the 21 states. The second direction is to build a map of as many FSSP solutions as possible and study how they relate through the notion of "reduction" introduced. Applications of this maps includes the discovery and systematisation of techniques used in handcrafted transition table and the factorisation of correctness proofs. The contributions of this thesis give answers to both directions and open new ones.

In Chapter 2, we first introduce the original results of Maignan and Yunes [15, 18], namely their infinite CA and its associated 21-state CA. Then, we show how to modify their approach to obtain less than 21 states from their infinite solution, but also how to apply the same procedure for another non-minimal-time FSSP algorithm by exposing a trade-off between the robustness of the reduced solution and its number of states. This is reminiscent of classical phenomenon in classical computer programming and exemplifies the high-level aspects claimed earlier. These results are not published. In this same chapter, we also show a first attempt to reduce the two finite CA at hand by quotienting their set of states. This does provide some improvements in the number of states and number of transitions, *e.g.*, the 21-state solution is reduced into to 14 states automatically. These results are published in [25] and [27].

The results of Chapter 3.6 only lead to 14 states, and can readily not give any better result based of the 21-state CA as explained in the conclusion of the chapter. In Chapter 3, we introduce a generalized kind of quotient which is able to go further. It allows the 21 states to be reduced to an 8-state solution, this 8-state CA being in fact a known solution.

in [25], [27]

and [26].

Also, inspired by the idea of *local search* and exploration through small modifications used in [3] to generate the 718 solutions, we tried such search algorithms to generate "reductions" of existing solutions rather than transition tables directly. Although the idea of local search is to navigate randomly in a landscape with few actual solutions, the discovered landscape of reductions has so many solutions that

a "best-effort-exhaustive" exploration have been tried, leading to *many millions* of 6-states solutions. Also, this space is much more easily explored because of its nice computational properties. These results are not specific to FSSP solutions. For example, considering the n^3-RTSG solutions and we provide millions of 5-state solutions and a 4-state solution to improve on the previous 6-state solution.In Chapter 2, we first introduce the original results of Maignan and Yunes [15, 18], namely their infinite CA and its associated 21-state CA.

Chapter 2

Field-Based FSSP and Quotients

We begin by recalling in Section 2.1 the required background about the field-based approach applied in [18] to solve the minimal-time FSSP. The adaptativeness of this approach has already been demonstrated at the algorithmic level in many articles solving different version of the FSSP (multidimensional GFSSP [19], hexagonal GFSSP [16], Mazoyer-like strategy adapted to GFSSP [17] and generalization to arbitrary ratio [20]). However, except for [18], the details are not given up to an actual cellular automaton transition table. So we proceed by demonstrating again the adaptativeness of this approach with full detail by rewiring the modules to match a different algorithm corresponding to non-minimal-time FSSP in Section 2.2. This is useful to exemplify some later concept too.

We then jump into the core of this thesis by considering ways to reduce the number of states of these FSSP solutions. We first do so by a slight modification of the previous procedures from infinite cellular automata: a modification of the reduction formulas. We then consider ways to improve on a given *finite* cellular automata, in a way analogous to what happen for deterministic finite automata: a simple quotient of the set of states. The limitations in this approach leads to the generalizations considered in the next chapter. Some of the results described in this chapter are published in [27].

2.1 Background on Fields for FSSP

Let us recall roughly the required background on the field-based approach to FSSP. For our concern, it is enough to see that, once all fields composed, the resulting cellular automaton is infinite, a state being a infinite tuple because of the recursive nature of the algorithm. We describe this infinite cellular automaton in Section 2.1.1. We then introduce the concept of family of space-time diagrams in Section 2.1.2, which is used to produce a finite cellular automaton out of the infinite one, as described in Section 2.1.3.

2.1.1 Infinite Cellular Automaton $\mathfrak{F}$

A detailed explanation of the field-based approach in out of the scope of this document, and is not required to follow its content. We refer the interested reader to [18] for more information. We nonetheless need to give a rough description of the resulting infinite cellular automaton formalizing conceptually the recursive splitting

of the space in halves, the most well-known algorithm to solve the FSSP as already shown in Figure 1.7.

A state of this infinite cellular automaton is composed of a first boolean `inp` , an infinite sequence of tuples, and a final boolean `out`. The three-valued `inp`, taking values in the set $\{\star, \top, \bot\}$, specifies which cells are outside of the finite configuration by $\star$, and which cells are allowed to start working according to the FSSP specification by $\top$, *i.e.* only the leftmost cell (the general) in the initial configuration, then any cell having a working neighbor. The boolean `out` specifies which cells fire. Each tuple of the infinite sequence encodes information for one level of splitting: the level 0 split the whole space in halves, the level 1 split the halves in quarters, and so on so forth. The tuple of a given level ℓ consists of 5 data: $\texttt{brd}^\ell$, $\texttt{ins}^\ell$, $\texttt{dst}^\ell$, $\texttt{sta}^\ell$, $\texttt{mid}^\ell$. The boolean $\texttt{brd}^\ell$ (respectively $\texttt{ins}^\ell$) is true when a cell is know to be a border (respectively a non-border) for the level ℓ. Two booleans are needed since a cell might not have determined yet its status, in which case both booleans are false. The natural number $\texttt{dst}^\ell$ indicates the lower bound distance of each cell to the nearest borders, up to current knowledge. The boolean $\texttt{sta}^\ell$ (for stability) is true when the distance is known to be correct because the knowledge will not evolve anymore. Finally, the boolean $\texttt{mid}^\ell$ (for middle) is true for the cells that are known to be at the splitting point of the current level of splitting.

In the initial configuration, all booleans are false, and all distance values are 0. In field `inp` is set to $\star$ outside of the finite configuration, to $\top$ at the general, and to $\bot$ at the other cells. The following configurations are obtained using the following evolution rules, where the field notation $f_t(c)$ is the value, in the space-time diagram, of the data/field f at time t and position/cell c. A detailed understanding of these formulas is not crucial.

$$\texttt{inp}_{t+1}(c) = \begin{cases} \star & \text{if } \texttt{inp}_t(c) = \star \\ \top & \text{if } \exists\, i \in \{-1,0,+1\};\, \texttt{inp}_t(c+i) = \top \\ \bot & \text{otherwise} \end{cases}$$

$$\texttt{brd}^0_{t+1}(c) = \texttt{inp}_{t+1}(c) = \top \wedge \exists\, i \in \{-1,+1\};\, \texttt{inp}_t(c+i) = \star.$$

$$\texttt{ins}^0_{t+1}(c) = \texttt{inp}_{t+1}(c) = \top \wedge \forall\, i \in \{-1,+1\};\, \texttt{inp}_t(c+i) \neq \star.$$

$$\texttt{dst}^\ell_{t+1}(c) = \begin{cases} min_{i\in\{-1,+1\}}\{1 + \texttt{dst}^\ell_t(c+i)\} & \text{if } \texttt{ins}^\ell_{t+1}(c) \\ 0 & \text{otherwise.} \end{cases}$$

$$\texttt{sta}^\ell_{t+1}(c) = \bigvee \begin{cases} \texttt{brd}^\ell_{t+1}(c) \\ \exists\, i \in \{-1,+1\};\, \texttt{dst}^\ell_{t+1}(c) = 1 + \texttt{dst}^\ell_t(c+i) \wedge\ \texttt{sta}^\ell_t(c+i). \end{cases}$$

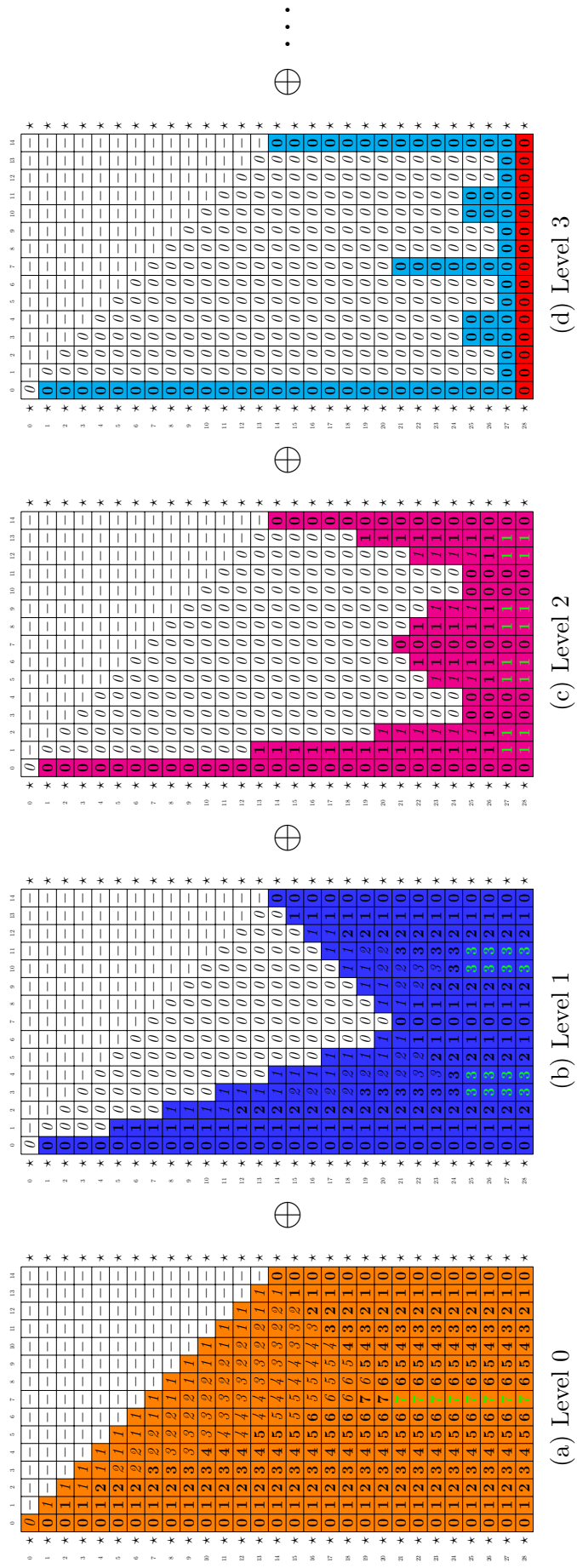
$$\texttt{mid}^\ell_{t+1}(c) = \bigvee \begin{cases} \texttt{dst}^\ell_{t+1}(c) > max_{i\in\{-1,+1\}}(\texttt{dst}^\ell_t(c+i)) \\ \qquad \wedge\, \forall\, i \in \{-1,+1\}, \texttt{sta}^\ell_t(c+i), \\ \texttt{dst}^\ell_{t+1}(c) = max_{i\in\{-1,+1\}}(\texttt{dst}^\ell_t(c+i)) \\ \qquad \wedge\, \forall\, i \in \{-1,0,+1\}, \texttt{sta}^\ell_t(c+i). \end{cases}$$

$$\texttt{brd}^{\ell+1}_{t+1}(c) = \texttt{brd}^\ell_{t+1}(c) \vee \texttt{mid}^\ell_{t+1}(c).$$

$$\texttt{ins}^{\ell+1}_{t+1}(c) = \texttt{ins}^\ell_{t+1}(c) \wedge \texttt{sta}^\ell_{t+1}(c) \wedge \exists\, i \in \{-1,+1\} : \texttt{dst}^\ell_t(c+i) > \texttt{dst}^\ell_{t+1}(c).$$

$$\texttt{out}_{t+1}(c) = \exists\, \ell \in \mathbb{N}, \forall\, i \in \{-1,0,+1\}\ ;\ \texttt{brd}^\ell_t(c+i).$$

Figure 2.1 shows the space-time diagram, level by level for readability. In each



(a) Level 0 (b) Level 1 (c) Level 2 (d) Level 3

Figure 2.1: Space-time diagram of the infinite CA $\mathfrak{F}$, by "levels of splitting".

level ℓ, $\texttt{brd}^\ell$ is true at cells containing a boldface 0, $\texttt{ins}^\ell$ is true at cells containing a non-null value, the value of $\texttt{dst}^\ell$ is the displayed value, $\texttt{sta}^\ell$ is true at cells with boldface values, and $\texttt{mid}^\ell$ is true at cells with highlighted values. In the level 3 of this example, all cells are ultimately identified as borders, at which point all cells fire according to the evolution rule of `out`. The main feature to look for in the following above formulas is that the initial level sees its data $\texttt{brd}^0$ and $\texttt{ins}^0$ built from `inp` only, while the following levels $\ell+1$, see their data $\texttt{brd}^{\ell+1}$ and $\texttt{ins}^{\ell+1}$ built from the data of the previous level ℓ only.

2.1.2 Cellular Automata and Family of Diagrams

In order to produce a finite cellular automaton from this infinite one, we need to make a detour and make precise the fact that a finite or infinite cellular automaton is a description of special family of space-time diagrams: deterministic ones. But we need to consider also more general, non-deterministic, families of space-time diagrams in some reasoning.

Definition 6. *A* family of space-time diagrams *D consists of a set of states Σ_D and an arbitrary set (abusively denoted) $D \subseteq {\Sigma_D}^{\mathbb{N}\times\mathbb{Z}}$ of space-time diagrams. The* (induced) local transition relation *$\delta_D \subseteq {\Sigma_D}^3 \times \Sigma_D$* of *$D$ is defined as:*

$$((c^0_{-1}, c^0_0, c^0_1), c^1_0) \in \delta_D :\Leftrightarrow \exists (d, t, p) \in D \times \mathbb{N} \times \mathbb{Z} \text{ s.t. } c^j_i = d(t + j, p + i).$$

We call D a deterministic family *if its induced local transition relation is functional.*

Just to grasp the concepts, note that two different (non-deterministic) families of space-time diagrams can have the same local transition relation. A given local transition relation have a maximal family of space-time diagrams. In this case, the local transition relation can be seen as a (partial) non-deterministic cellular automaton, and the maximal family is just the family of all possible (complete) diagrams. When the family is deterministic, one have the following.

Definition 7. *Given a deterministic family D, its* associated cellular automaton *Γ_D is defined as having the set of states $\Sigma_{\Gamma_D} = \Sigma_D$, the set of initial configurations $\mathrm{I}_{\Gamma_D} = \{d(0, -) \in {\Sigma_\alpha}^{\mathbb{Z}} \mid d \in D\}$, and the local transition function $\delta_{\Gamma_D} = \delta_D$.*

Definition 8. *Given a cellular automaton α, its* associated family of space-time diagrams *(abusively denoted) D_α is defined as having the set of states $\Sigma_{\mathrm{D}_\alpha} = \Sigma_\alpha$, and the set of space-time diagrams $\{\, \mathrm{D}_\alpha(c) \mid c \in \mathrm{I}_\alpha \,\}$ and is clearly deterministic.*

These inverse constructions show that deterministic families and cellular automata are two presentations of the same object. For practical purposes, it is also useful to note that, since δ_D has a finite domain, there are finite subsets of D that are enough to specify it completely.

2.1.3 The Finite Cellular Automaton $\mathfrak{F}^{21}_{486}$

Now we have everything to produce a finite cellular automaton solving the FSSP. The idea is to program the infinite cellular automaton, to generate an appropriate number of space-time diagrams from it, to modify the states in these space-time

diagrams, and to extract the local transition relation from these modified space-time diagrams, hoping for it to be functional.

The modification applied on the states have to extract a finite information out of the unbounded sequence of tuples, but to conserve enough information to obtain a deterministic family of space-time diagram, and therefore reconstruct a cellular automaton at the end. The following extraction formulas, also called reduction formulas, are used. Note that they do not describe an evolution, just an extraction of information.

$$\texttt{rbrd} = \texttt{inp} \wedge \exists\, \ell \in \mathbb{N}\ ;\ \texttt{dst}^{\ell} = 0 \wedge \texttt{sta}^{\ell}.$$

$$\texttt{lvl} = \begin{cases} 0 & \text{if } \texttt{rbrd} \vee \neg\ \texttt{inp}; \\ \min\{\ell \in \mathbb{N} \mid \texttt{dst}^{\ell+1} = 0\} & \text{otherwise.} \end{cases}$$

$$\texttt{rlvl} = \texttt{lvl} \bmod 3.$$

$$\texttt{rdst} = \begin{cases} 0 & \text{if } \texttt{rbrd} \vee \neg\ \texttt{inp}; \\ \texttt{dst}^{\texttt{lvl}} \bmod 3 & \text{otherwise.} \end{cases}$$

$$\texttt{rsta} = \texttt{sta}^{\texttt{lvl}} \vee \texttt{rbrd}.$$

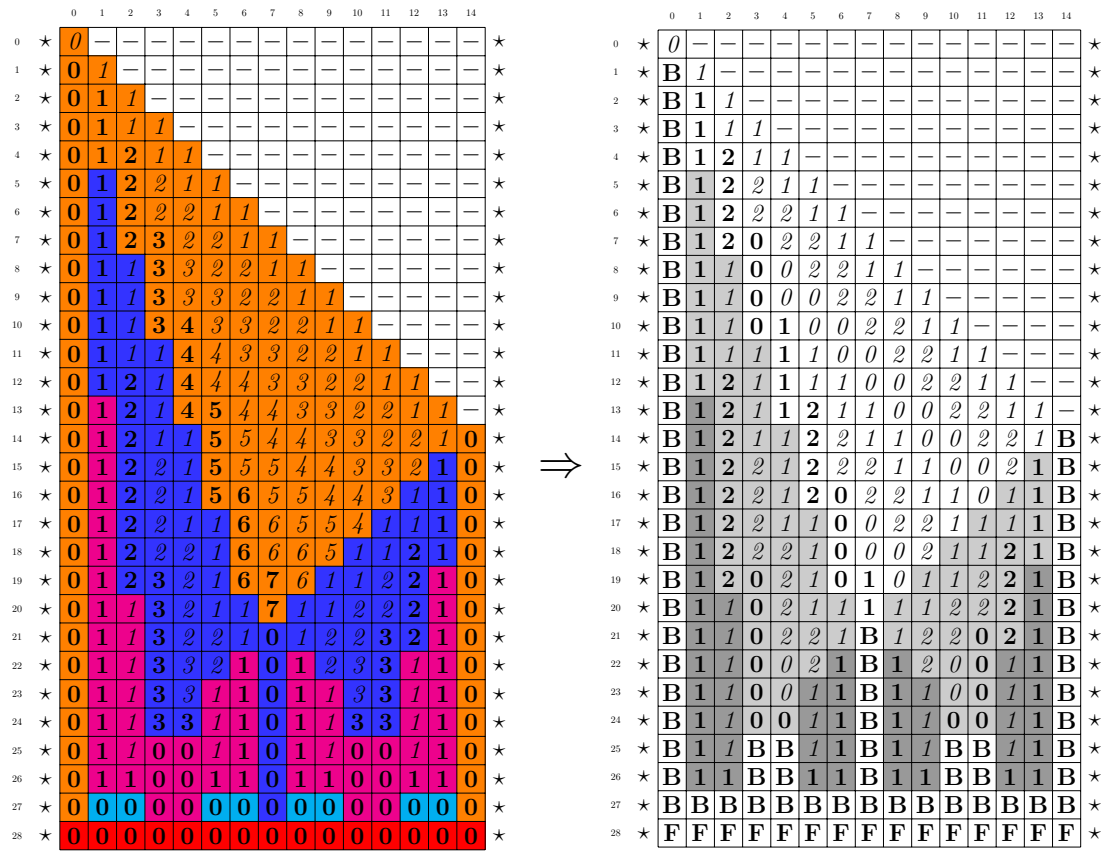
In fact, the resulting finite state σ is a tuple of six data $\texttt{inp}_\sigma$, $\texttt{rlvl}_\sigma$, $\texttt{rdst}_\sigma$, $\texttt{rsta}_\sigma$, $\texttt{rbrd}_\sigma$, $\texttt{out}_\sigma$. The semantic of $\texttt{inp}_\sigma$ and $\texttt{out}_\sigma$ is already described. The other data correspond the relevant core part of the active level $\texttt{lvl}$, the distance value $\texttt{dst}^{\texttt{lvl}}$ being moded out into $\texttt{rdst}_\sigma$, and the values of $\texttt{lvl}$ too in $\texttt{rlvl}_\sigma$.

Figure 2.2 shows a summary of an original space-time diagram, and the resulting finite-state diagram. In this resulting diagram, the value $\texttt{inp}_\sigma$ is false only for the quiescent state ▫. The value of $\texttt{rlvl}_\sigma$ is represented by the background color : 0 is □, 1 is ▒ and 2 is ■. The numeric value of $\texttt{rdst}_\sigma$ is indicated directly in the state and this value is 0 when no value are indicated. The value of $\texttt{rsta}_\sigma$ is true for states with an bold digit, *e.g.* 1, and for B, F and false for states with an italic digit, *e.g.* *1*, and for ▫. The value of $\texttt{rbrd}_\sigma$ is true only for B and F and $\texttt{rout}_\sigma$ is true only for F.

Following the procedure, we generate a finite subpart of its family of space-time diagrams, i.e. the space-time diagrams associated with the FSSP initial configuration of size 2 to 1000. For each of space-time diagram, we transformed each state according to the prescribed reduction formulas to produce a new family of space-time diagram. This family being appropriately deterministic, we extracted the induced local transition table of the cellular automaton associated to it. In fact, the extraction was already complete with the FSSP initial configurations of size 2 to 105.

The result is a cellular automaton of 21 states and 486 local transition rules that we denote $\mathfrak{F}^{21}_{486}$. Among these rules, there are 477 symmetric rules consisting of 23 self-symmetric rules and 227 pairs of symmetric rule. A rule $(a, b, c) \mapsto d \in \delta_{\mathfrak{F}^{21}_{486}}$ is self-symmetric if $a = c$ and symmetric if $(c, b, a) \mapsto d \in \delta_{\mathfrak{F}^{21}_{486}}$. In Figure 2.3 in page 27 is depicted the 9 asymmetric rules firsts, and then the symmetric rules, keeping only one element of each pair of (non-self-)symmetric rules.

Note that, in fact, the original cellular automaton $\mathfrak{F}$ is not just a solution for the FSSP, but for the asynchronous multi general FSSP, and even for a novel kind of FSSP that we call "full FSSP" where each cell decides to become non-quiescent independently from the others. This is a side remark to explain partially why it is



(a) Summary of the original space-time diagram of $\mathfrak{F}$ depicted in Figure 2.1

(b) Corresponding extracted finite-state diagram leading to finite CA $\mathfrak{F}^{21}_{486}$

Figure 2.2: Field-based (finite) CA $\mathfrak{F}^{21}_{486}$ extracted from infinite CA $\mathfrak{F}$

not surprising to obtain more state than usual. However, it should be possible to optimize the number of states and the number of transitions to a competitive level, which is one of the motivation of this work as already discussed.

2.2 Demonstrating Field-Based Modularity: $\mathfrak{F}'$

The previous solution correspond to the FSSP algorithm consisting in the recursive splitting in halves in minimal time $2n-2$. Let us now apply the same procedure for a slightly different FSSP solution algorithm. The goal is to match the $3n+O(\log n)$ algorithm depicted in Figure 1.8b to demonstrate the modularity of the field-based approach, but also to prepare for some later investigation.

The infinite CA encoding this $3n + O(\log n)$ algorithm is obtained by only a slight modification of the previous evolution rules. Indeed, thinking in terms of fields, the idea of this algorithm is still to split recursively the space in halves. The only difference is that each level of splitting works independently, a large part of the useful information built at each level being discarded by the next level to only keep the splitting. Moreover, at each change of level, a delay is introduced for finite-state encoding reasons. To capture these two features, it is enough to add to each level $\ell+1$ an additional field/data $\texttt{inp}^{\ell+1}$ that simply propagates at speed 1 from the splitting of level ℓ, in the same way as the original `inp` field of the previous

—	—	⋆	→	—	*0*	—	—	→	*1*	*1*	—	—	→	*1*	**1**	*1*	—	→	*1*	B	*1*	—	→	**1**
0	—	⋆	→	B	*1*	—	⋆	→	B	⋆	*0*	—	→	B	*1*	*1*	—	→	*1*					

(a) Asymmetric rules

—	—	—	→	—	*0*	**1**	*1*	→	**1**	*1*	*1*	*1*	→	*1*	*1*	*1*	**2**	→	*1*	*0*	*2*	*2*	→	*0*
0	**0**	*1*	→	B	**1**	**1**	*1*	→	B	*1*	*1*	**2**	→	*2*	*1*	*1*	*2*	→	*1*	**1**	*2*	**2**	→	**2**
0	**0**	*1*	→	**0**	**1**	**1**	B	→	B	*1*	*1*	**2**	→	*1*	*1*	*1*	*1*	→	*1*	*1*	*2*	**2**	→	*2*
1	**0**	*1*	→	*1*	*1*	**1**	*1*	→	**1**	*1*	*1*	*2*	→	*2*	*1*	*1*	**2**	→	*1*	*1*	*2*	*2*	→	*2*
1	**0**	*1*	→	B	*1*	**1**	B	→	**1**	*1*	*1*	*2*	→	*1*	*1*	*1*	**2**	→	*2*	*1*	*2*	**2**	→	**2**
1	**0**	*2*	→	**0**	*1*	**1**	*1*	→	B	*1*	*1*	**2**	→	*1*	*1*	*1*	*2*	→	*1*	**2**	*2*	*2*	→	**0**
2	**0**	*2*	→	**0**	*1*	**1**	**2**	→	*1*	*1*	*1*	*2*	→	*1*	*1*	*1*	*2*	→	*2*	**0**	*2*	**1**	→	*1*
0	**0**	*1*	→	B	**2**	**1**	B	→	**1**	**2**	*1*	B	→	**1**	**2**	*1*	*2*	→	*1*	**0**	*2*	*1*	→	*2*
0	**0**	*1*	→	**0**	**2**	**1**	B	→	**1**	**2**	*1*	*2*	→	*1*	**2**	*1*	B	→	**1**	**0**	*2*	*1*	→	*1*
1	**0**	*1*	→	*1*	*2*	**1**	B	→	**1**	*2*	*1*	B	→	**1**	*2*	*1*	B	→	**1**	**0**	*2*	*2*	→	*0*
1	**0**	*1*	→	B	B	**1**	B	→	B	**0**	*1*	**1**	→	*1*	**0**	**2**	**1**	→	*1*	*0*	*2*	**1**	→	*1*
1	**0**	*2*	→	**0**	**0**	**1**	*0*	→	**1**	**0**	*1*	*1*	→	*1*	**0**	**2**	*1*	→	*1*	*0*	*2*	*1*	→	*2*
2	**0**	*2*	→	**0**	*0*	**1**	*1*	→	**1**	**0**	*1*	*2*	→	*1*	**1**	**2**	*1*	→	**2**	*0*	*2*	*1*	→	*1*
0	**0**	*1*	→	B	**1**	**1**	*1*	→	B	*0*	*1*	**1**	→	*1*	**1**	**2**	**1**	→	B	*0*	*2*	*2*	→	*0*
0	**0**	*1*	→	**0**	**1**	**1**	B	→	B	*0*	*1*	*1*	→	*1*	**1**	**2**	*1*	→	**2**	**1**	*2*	**2**	→	**2**
1	**0**	*1*	→	*1*	*1*	**1**	*1*	→	**1**	*0*	*1*	**1**	→	*1*	**1**	**2**	**2**	→	B	*1*	*2*	**2**	→	*2*
1	**0**	*1*	→	B	*1*	**1**	B	→	**1**	*0*	*1*	*1*	→	*1*	**1**	**2**	*2*	→	**2**	*1*	*2*	*2*	→	*2*
1	**0**	*2*	→	**0**	*1*	**1**	*1*	→	B	*0*	*1*	**2**	→	*1*	*1*	**2**	*1*	→	**2**	*1*	*2*	**2**	→	**2**
2	**0**	*2*	→	**0**	*1*	**1**	**2**	→	*1*	*0*	*1*	*2*	→	*1*	*1*	**2**	*1*	→	B	**2**	*2*	*2*	→	**0**
0	*0*	*0*	→	**1**	**2**	**1**	B	→	**1**	**1**	*1*	*1*	→	*1*	*1*	**2**	**2**	→	B	**0**	*2*	**1**	→	*1*
0	*0*	*1*	→	**0**	**2**	**1**	B	→	**1**	**1**	*1*	**2**	→	*1*	*1*	**2**	*2*	→	**2**	**0**	*2*	*1*	→	*1*
0	*0*	*2*	→	*0*	*2*	**1**	B	→	**1**	**1**	*1*	*2*	→	*1*	**0**	**2**	**1**	→	*1*	**0**	*2*	*1*	→	*2*
0	*0*	**1**	→	*1*	B	**1**	B	→	B	**1**	*1*	*1*	→	**2**	**0**	**2**	*1*	→	*1*	**0**	*2*	*2*	→	*0*
0	*0*	*1*	→	*1*	**0**	**1**	*0*	→	**1**	**1**	*1*	*1*	→	**1**	**1**	**2**	*1*	→	**2**	*0*	*2*	**1**	→	*1*
0	*0*	*2*	→	*0*	*0*	**1**	*1*	→	**1**	**1**	*1*	B	→	**1**	**1**	**2**	**1**	→	B	*0*	*2*	*1*	→	*1*
1	*0*	*1*	→	*1*	**1**	**1**	*1*	→	B	*1*	*1*	*1*	→	*1*	**1**	**2**	*1*	→	**2**	*0*	*2*	*1*	→	*2*
1	*0*	*2*	→	*0*	**1**	**1**	B	→	B	*1*	*1*	**2**	→	*1*	**1**	**2**	**2**	→	B	*0*	*2*	*2*	→	*0*
1	*0*	*1*	→	*1*	*1*	**1**	*1*	→	B	*1*	*1*	**2**	→	*2*	**1**	**2**	*2*	→	**2**	**1**	*2*	**2**	→	**2**
1	*0*	*2*	→	*0*	*1*	**1**	*1*	→	**1**	*1*	*1*	*2*	→	*1*	*1*	**2**	*1*	→	**2**	*1*	*2*	**2**	→	**2**
0	*0*	*0*	→	**1**	*1*	**1**	**2**	→	*1*	*1*	*1*	*2*	→	*2*	*1*	**2**	*1*	→	B	*1*	*2*	**2**	→	*2*
0	*0*	*1*	→	**0**	*1*	**1**	B	→	**1**	*1*	*1*	**2**	→	*1*	*1*	**2**	**2**	→	B	*1*	*2*	*2*	→	*2*
0	*0*	*2*	→	*0*	**2**	**1**	B	→	**1**	*1*	*1*	*2*	→	*1*	*1*	**2**	*2*	→	**2**	**2**	*2*	*2*	→	**0**
0	*0*	**1**	→	*1*	**2**	**1**	B	→	**1**	**2**	*1*	*2*	→	*1*	**0**	**2**	**1**	→	*1*	⋆	B	**1**	→	B
0	*0*	*1*	→	*1*	*2*	**1**	B	→	**1**	**2**	*1*	B	→	**1**	**0**	**2**	*1*	→	*1*	⋆	B	**1**	→	B
0	*0*	*2*	→	*0*	B	**1**	B	→	B	*2*	*1*	B	→	**1**	**1**	**2**	**1**	→	B	⋆	B	**1**	→	B
1	*0*	*1*	→	*1*	**0**	*1*	**1**	→	*1*	**0**	*1*	**1**	→	*1*	**1**	**2**	*1*	→	**2**	⋆	B	*1*	→	B
1	*0*	*2*	→	*0*	**0**	*1*	*1*	→	*1*	**0**	*1*	*1*	→	*1*	**1**	**2**	**2**	→	B	⋆	B	B	→	**F**
1	*0*	*1*	→	*1*	**0**	*1*	*2*	→	*1*	**0**	*1*	*2*	→	*1*	**1**	**2**	*2*	→	**2**	**1**	B	**1**	→	B
1	*0*	*2*	→	*0*	*0*	*1*	**1**	→	*1*	*0*	*1*	**1**	→	*1*	**1**	**2**	*1*	→	**2**	**1**	B	B	→	B
0	*0*	*0*	→	**1**	*0*	*1*	*1*	→	*1*	*0*	*1*	*1*	→	*1*	*1*	**2**	*1*	→	B	**1**	B	**1**	→	B
0	*0*	*1*	→	**0**	*0*	*1*	**2**	→	*1*	*0*	*1*	**1**	→	*1*	*1*	**2**	*1*	→	**2**	**1**	B	B	→	B
0	*0*	*2*	→	*0*	*0*	*1*	*2*	→	*1*	*0*	*1*	*1*	→	*1*	*1*	**2**	**2**	→	B	**1**	B	**1**	→	B
0	*0*	**1**	→	*1*	*0*	*1*	**1**	→	*1*	*0*	*1*	**2**	→	*1*	*1*	**2**	*2*	→	**2**	**1**	B	B	→	B
0	*0*	*1*	→	*1*	*0*	*1*	*1*	→	*1*	*0*	*1*	*2*	→	*1*	**0**	*2*	**1**	→	*1*	*1*	B	*1*	→	B
0	*0*	*2*	→	*0*	**1**	*1*	*1*	→	**2**	**1**	*1*	*1*	→	*1*	**0**	*2*	*1*	→	*2*	*1*	B	B	→	B
1	*0*	*1*	→	*1*	**1**	*1*	*1*	→	**1**	**1**	*1*	**2**	→	*1*	**0**	*2*	*1*	→	*1*	*1*	B	*1*	→	B
1	*0*	*2*	→	*0*	**1**	*1*	B	→	**1**	**1**	*1*	*2*	→	*1*	**0**	*2*	*2*	→	*0*	*1*	B	B	→	B
1	*0*	*1*	→	*1*	**1**	*1*	*1*	→	*1*	**1**	*1*	*1*	→	**1**	*0*	*2*	**1**	→	*1*	*1*	B	*1*	→	B
1	*0*	*2*	→	*0*	**1**	*1*	**2**	→	*1*	**1**	*1*	*1*	→	**2**	*0*	*2*	*1*	→	*2*	*1*	B	B	→	B
0	**1**	*0*	→	**1**	**1**	*1*	*2*	→	*1*	**1**	*1*	B	→	**1**	*0*	*2*	*1*	→	*1*	B	B	B	→	**F**

(b) Symmetric rules

Figure 2.3: Transition table of Maignan - Yunès's solution $\mathfrak{F}^{21}_{486}$

field-based solution propagated at speed 1 from the leftmost cell. This latter field is renamed $\texttt{inp}^0$ and the clever computation of $\texttt{ins}^{\ell+1}$ and $\texttt{brd}^{\ell+1}$ – that used as much information as possible from the level ℓ in the previous solution – is replaced by a computation only based on the propagation on $\texttt{inp}^{\ell+1}$ with a delay.

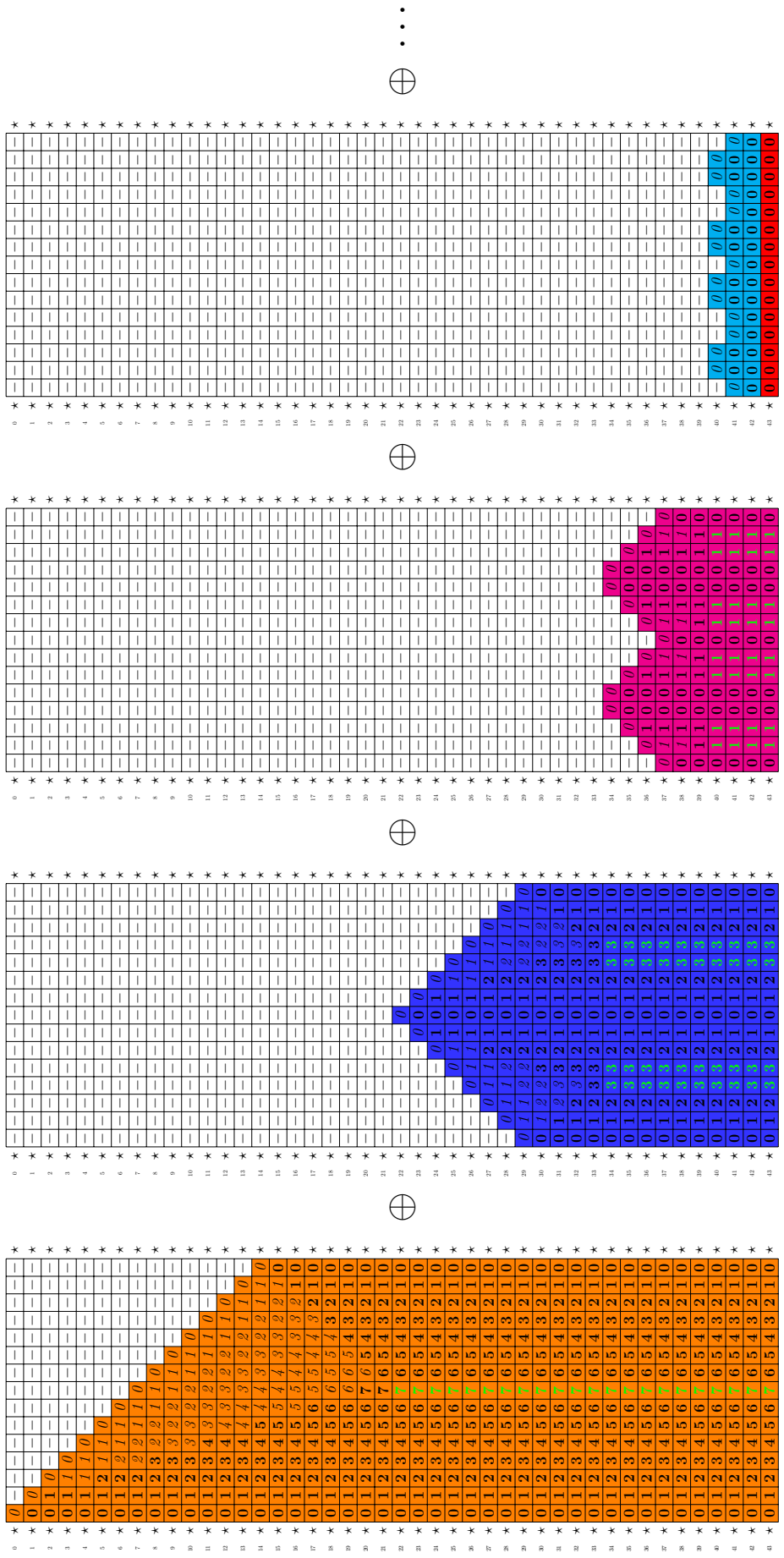
$$\begin{aligned}
\texttt{inp}^{\ell+1}_{t+1}(p) &= \texttt{mid}^{\ell}_{t+1}(p) \vee \exists\, i \in \{-1,0,1\};\ \texttt{inp}^{\ell+1}_{t}(p+i).\\
\texttt{brd}^{\ell+1}_{t+1}(p) &= \texttt{inp}^{\ell+1}_{t}(p) \wedge (\texttt{brd}^{\ell}_{t}(p) \vee \texttt{mid}^{\ell}_{t}(p)).\\
\texttt{ins}^{\ell+1}_{t+1}(p) &= \texttt{inp}^{\ell+1}_{t}(p) \wedge \neg\, \texttt{brd}^{\ell+1}_{t+1}(p).
\end{aligned}$$

The rules specify that when middles are detected at a level, the next level start from scratch with its own $\texttt{inp}^{\ell+1}$, $\texttt{brd}^{\ell+1}$ and $\texttt{ins}^{\ell+1}$ being built naively from $\texttt{inp}^{\ell+1}$ and the borders and middles of the previous level, with a slight delay $t \to t+1$ in the evolution rule of $\texttt{brd}^{\ell+1}$. Let us call this infinite cellular automaton $\mathfrak{F}'$. Figure 2.4 shows the impact of these modifications on the space-time diagrams, that should be compared to Figure 2.1. A summary of the differences is the previously solution launched all level simultaneously, the information between the levels being pipelined as soon as possible, while this solution launches the levels one after the other, in a kind of sequential manner.

The next step is the process of building a finite-state solution. For that, we use of the following reduction formulas and extract, from each infinite state, a finite-information tuple composed of 7 data `rinp`, `rlvl`, `rchlvl`, `rbrd`, `rdst`, `rsta` and `out`.

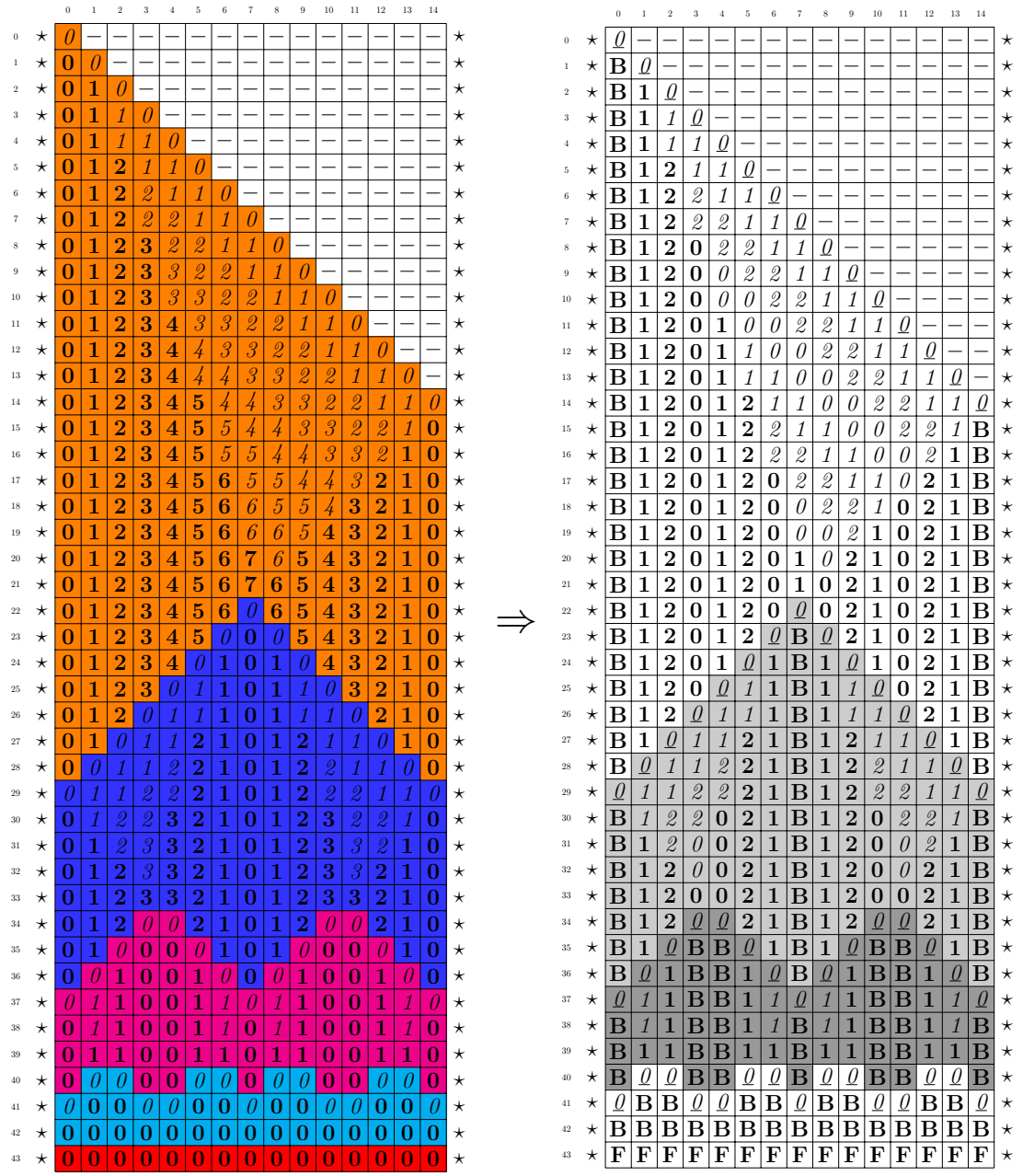
$$\begin{aligned}
\texttt{lvl} &= \begin{cases} 0 & \text{if } \neg\, \texttt{inp}^0 \\ max\{\ell \in \mathbb{N} \mid \texttt{inp}^{\ell}\} & \text{otherwise.} \end{cases}\\
\texttt{rinp} &= \texttt{inp}^{\texttt{lvl}}\\
\texttt{rlvl} &= \begin{cases} 0 & \text{if } \texttt{rout} \\ \texttt{lvl} \bmod 3 & \text{otherwise.} \end{cases}\\
\texttt{rchlvl} &= \texttt{inp}^{\texttt{lvl}} \wedge \texttt{dst}^{\texttt{lvl}} = 0 \wedge \neg\, \texttt{sta}^{\texttt{lvl}}.\\
\texttt{rbrd} &= \texttt{brd}^{\texttt{lvl}}.\\
\texttt{rdst} &= \begin{cases} 0 & \text{if } \texttt{brd}^{\texttt{lvl}} \\ \texttt{dst}^{\texttt{lvl}} \bmod 3 & \text{otherwise.} \end{cases}\\
\texttt{rsta} &= \texttt{sta}^{\texttt{lvl}}.
\end{aligned}$$

Doing so, we obtain, for instance, the finite-state space-time diagram depicted in Figure 2.5. The family of space-time diagram is deterministic of course, and the extracted cellular automaton has 26 states and 555 transition rules, noted $\mathfrak{F}'^{26}_{555}$. The encoding of the states is has before, with the additional field `rchlvl` being true at cells with an underlined 0. The transition table has been extracted and checked for determinism with FSSP initial configuration of size 2 to 1000 but the transition table is already complete with the initial configuration up to size 89. The resulting transition table is given in Figure 2.6.



(a) Level 0 (b) Level 1 (c) Level 2 (d) Level 3

Figure 2.4: Space-time diagram of the infinite CA $\mathfrak{F}'$, by "levels of splitting".



(a) Summary of the original space-time diagram of $\mathfrak{F}'$ depicted in Figure 2.4

(b) Corresponding extracted finite-state diagram leading to finite CA $\mathfrak{F}'^{26}_{555}$

Figure 2.5: Field-based (finite) CA $\mathfrak{F}'^{26}_{555}$ extracted from infinite CA $\mathfrak{F}'$

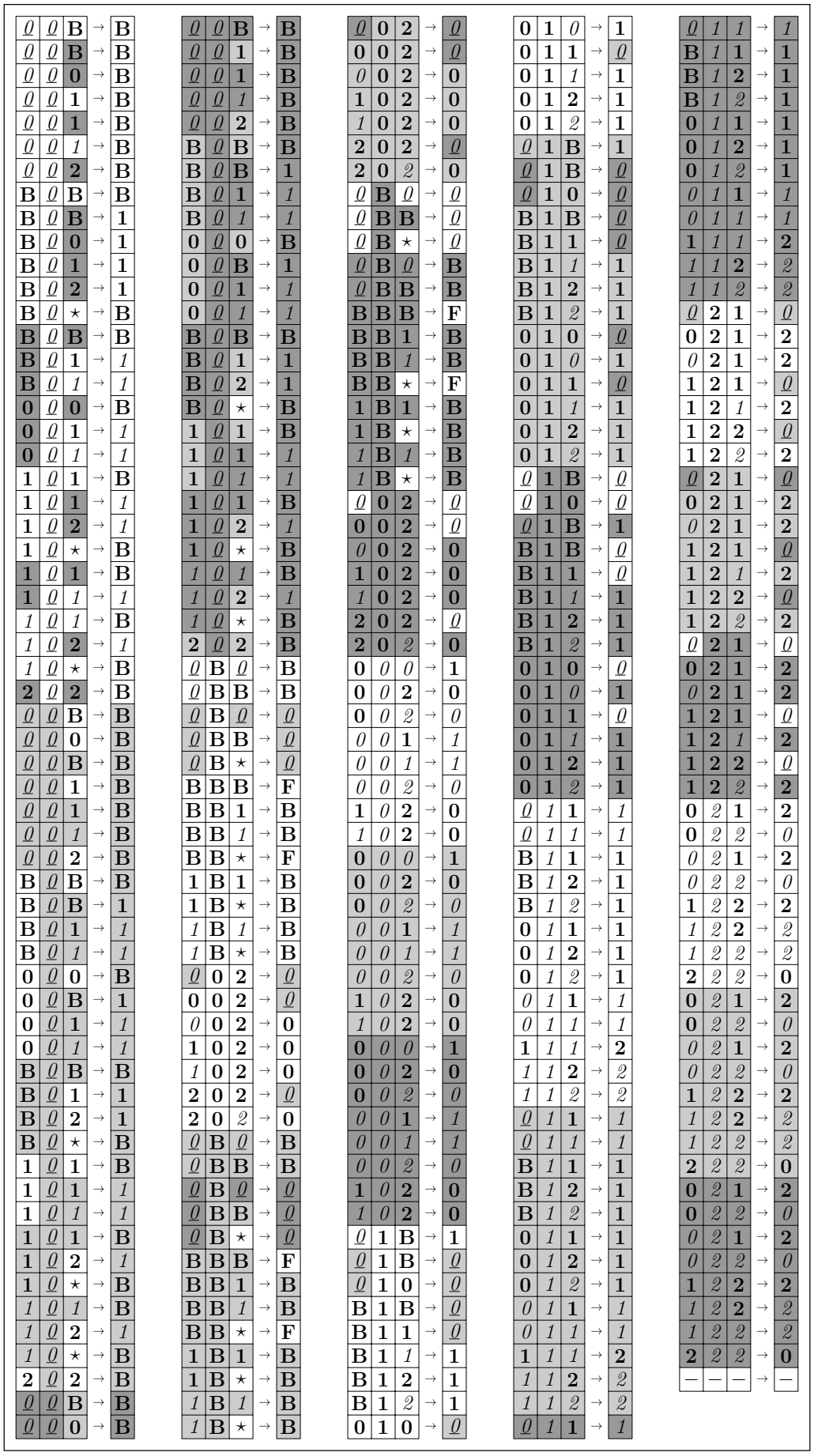
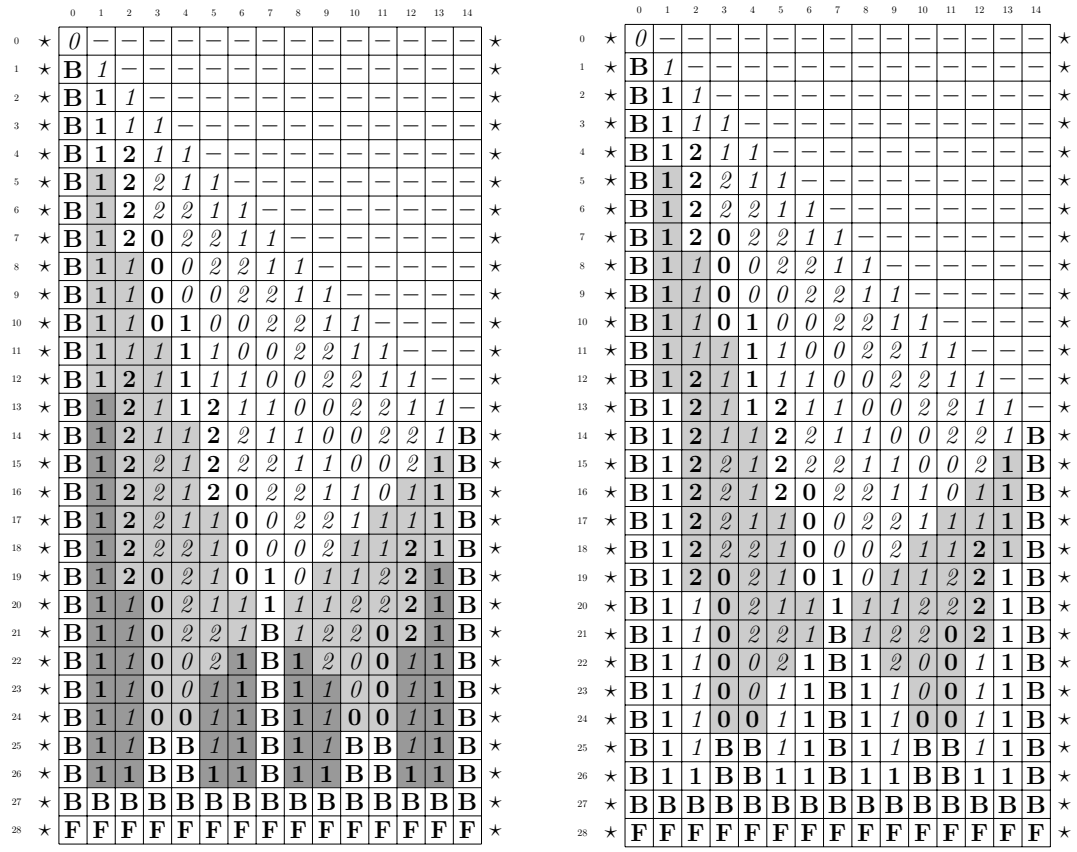


Figure 2.6: Symmetric part of transition table of $\mathfrak{F}'^{26}_{555}$. The asymmetric part is the same the CA $\mathfrak{F}'^{18}_{373}$ as given in Figure 2.9a.



(a) $\mathfrak{F}^{21}_{486}$ using `lvl` mod 3

(b) $\mathfrak{F}^{15}_{329}$ using `lvl` mod 2

Figure 2.7: Space-time diagram of size 15 of $\mathfrak{F}^{21}_{486}$ and $\mathfrak{F}^{15}_{329}$.

2.3 Optimizations of Field-Based Solutions

Our goal is to find ways to optimize the finite CA obtained from the field-based infinite CA. We begin by doing so by hand, at the level of the reduction formulas. We then consider an exhaustive exploration of quotients of the finite CA obtained with the reductions formulas. We then summarizing the results obtained so far and look at the limitations of the approach to overcome them.

2.3.1 Improving the Reduction Formulas from $\mathfrak{F}$ and $\mathfrak{F}'$

When extracting a finite field-based solution from infinite field-based solution, the number of states of the finite field-based solution depends on two "modulo" formulas:

$$\texttt{rdst} = \begin{cases} 0 & \text{if } \texttt{rbrd} \vee \neg\ \texttt{inp}; \\ \texttt{dst}^{\texttt{lvl}} \bmod 3 & \text{otherwise.} \end{cases}$$

$$\texttt{rlvl} = \texttt{lvl} \bmod 3.$$

It is very tempting to try to obtain less state by using smaller values for these modulo.

The first modulo is intrinsic to the fact that, given two neighboring cells, and fix the distance value d of the left cell for instance, the right can take value $d-1$, d or $d+1$ and all three cases happen in a complex way. Indeed, replacing this modulo

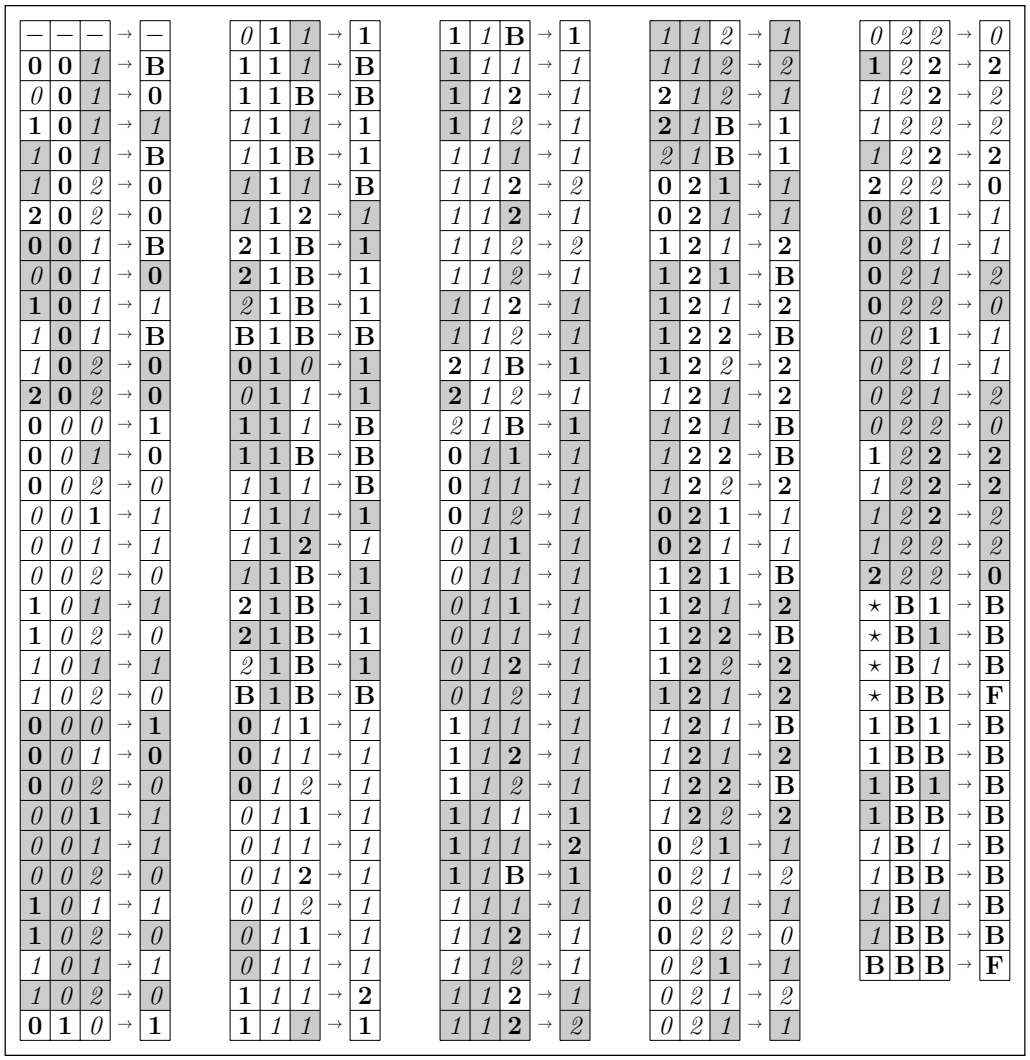


Figure 2.8: Symmetric rules of the transition table of $\mathfrak{F}^{15}_{329}$. The asymmetric part is the same the CA $\mathfrak{F}^{21}_{486}$ as given in Figure 2.3

.

3 by modulo 2 and repeating the finite state CA generation process, we found that the family of space-time diagram obtained in this way is not deterministic.

The second modulo is 3 for the same reason as the first one, but there is more structure, so we tried to replace this modulo 3 by modulo 2 in the reductions formulas and repeated the process of generating a finite state CA. This time, the family of space-time diagrams is deterministic (size 2 to 1000 as before, the transition table being complete at size 53). This CA is noted $\mathfrak{F}^{15}_{329}$ and has 15 states, 329 rules. Figure 2.7b shows its space-time diagram for size 15 and Figure 2.8 shows its local transition table.

In the same way, it is possible to change this second modulo from 3 to 2 in the formulas `rlvl` of the non-minimal-time infinite solution $\mathfrak{F}'$. The result is a deterministic family of diagrams, and CA $\mathfrak{F}'^{18}_{373}$ which has 18 states and 373 rules (built from size 2 to 1000 as before, the table being complete at size 45). Figure 2.10a shows a space-time diagram of size 15 and Figure 2.9 shows the local transition table of $\mathfrak{F}'^{18}_{373}$.

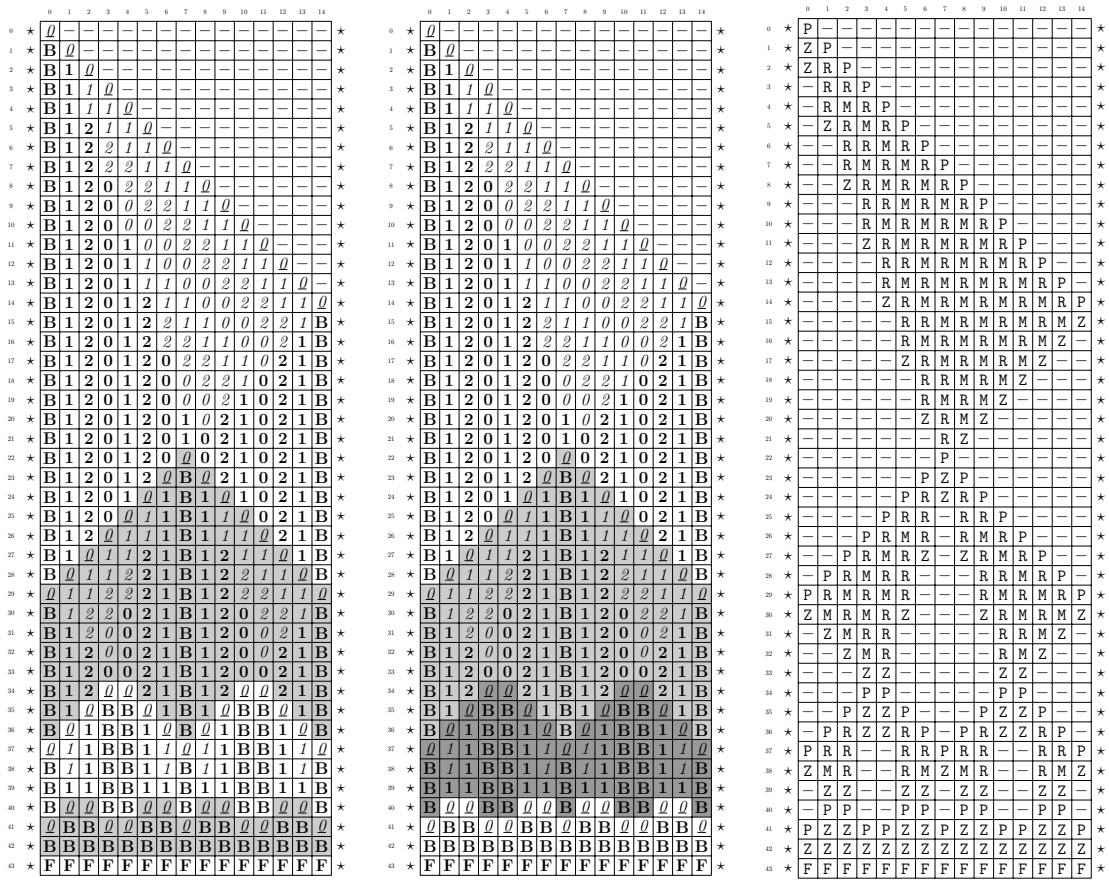
B	*0*	—	→	1	*1*	*0*	—	→	*1*	⋆	B	*0*	→	B	*0*	—	—	→	*0*
1	*0*	—	→	*1*	⋆	*0*	—	→	B	*0*	—	⋆	→	*0*	—	—	⋆	→	—

(a) Asymmetric rules

0	*0*	B	→	B	0	*0*	0	→	B	*0*	B	⋆	→	*0*	B	1	2	→	1	0	*1*	1	→	1
0	*0*	B	→	B	0	*0*	B	→	1	*0*	B	*0*	→	B	B	1	*2*	→	1	0	*1*	2	→	1
0	*0*	0	→	B	0	*0*	1	→	*1*	*0*	B	B	→	B	0	1	0	→	*0*	0	*1*	*2*	→	1
0	*0*	1	→	B	0	*0*	*1*	→	*1*	B	B	B	→	F	0	1	*0*	→	1	*0*	*1*	1	→	*1*
0	*0*	1	→	B	B	*0*	B	→	B	B	B	1	→	B	0	1	1	→	*0*	*0*	*1*	*1*	→	*1*
0	*0*	*1*	→	B	B	*0*	1	→	1	B	B	*1*	→	B	0	1	*1*	→	1	1	*1*	*1*	→	2
0	*0*	2	→	B	B	*0*	2	→	1	B	B	⋆	→	F	0	1	2	→	1	*1*	*1*	2	→	*2*
B	*0*	B	→	B	B	*0*	⋆	→	B	1	B	1	→	B	0	1	*2*	→	1	*1*	*1*	*2*	→	*2*
B	*0*	B	→	1	1	*0*	1	→	B	1	B	⋆	→	B	*0*	1	B	→	*0*	*0*	2	1	→	*0*
B	*0*	0	→	1	1	*0*	1	→	*1*	*1*	B	*1*	→	B	*0*	1	0	→	*0*	0	2	1	→	2
B	*0*	1	→	1	1	*0*	*1*	→	*1*	*1*	B	⋆	→	B	*0*	1	B	→	1	*0*	2	1	→	2
B	*0*	2	→	1	1	*0*	1	→	B	*0*	0	2	→	*0*	B	1	B	→	*0*	1	2	1	→	*0*
B	*0*	⋆	→	B	1	*0*	2	→	*1*	0	0	2	→	*0*	B	1	1	→	*0*	1	2	*1*	→	2
B	*0*	B	→	B	1	*0*	⋆	→	B	*0*	0	2	→	0	B	1	*1*	→	1	1	2	2	→	*0*
B	*0*	1	→	*1*	*1*	*0*	*1*	→	B	1	0	2	→	0	B	1	2	→	1	1	2	*2*	→	2
B	*0*	*1*	→	*1*	*1*	*0*	2	→	*1*	*1*	0	2	→	0	B	1	*2*	→	1	*0*	2	1	→	*0*
0	*0*	0	→	B	*1*	*0*	⋆	→	B	2	0	2	→	*0*	0	1	0	→	*0*	0	2	1	→	2
0	*0*	1	→	*1*	2	*0*	2	→	B	2	0	*2*	→	0	0	1	*0*	→	1	*0*	2	1	→	2
0	*0*	*1*	→	*1*	*0*	B	*0*	→	B	0	*0*	*0*	→	1	0	1	1	→	*0*	1	2	1	→	*0*
1	*0*	1	→	B	*0*	B	B	→	B	0	*0*	2	→	0	0	1	*1*	→	1	1	2	*1*	→	2
1	*0*	1	→	*1*	*0*	B	*0*	→	*0*	0	*0*	*2*	→	*0*	0	1	2	→	1	1	2	2	→	*0*
1	*0*	2	→	*1*	*0*	B	B	→	*0*	*0*	*0*	1	→	*1*	0	1	*2*	→	1	1	2	*2*	→	2
1	*0*	⋆	→	B	*0*	B	⋆	→	*0*	*0*	*0*	*1*	→	*1*	*0*	*1*	1	→	*1*	0	*2*	1	→	2
1	*0*	1	→	B	B	B	B	→	F	*0*	*0*	*2*	→	*0*	*0*	*1*	*1*	→	*1*	0	*2*	*2*	→	*0*
1	*0*	*1*	→	*1*	B	B	1	→	B	1	*0*	2	→	0	B	*1*	1	→	1	*0*	*2*	1	→	2
1	*0*	*1*	→	B	B	B	*1*	→	B	*1*	*0*	2	→	0	B	*1*	2	→	1	*0*	*2*	*2*	→	*0*
1	*0*	2	→	*1*	B	B	⋆	→	F	0	*0*	*0*	→	1	B	*1*	*2*	→	1	1	*2*	2	→	2
1	*0*	⋆	→	B	1	B	1	→	B	0	*0*	2	→	0	0	*1*	1	→	1	*1*	*2*	2	→	*2*
2	*0*	2	→	B	1	B	⋆	→	B	0	*0*	*2*	→	*0*	0	*1*	2	→	1	*1*	*2*	*2*	→	*2*
0	*0*	B	→	B	*1*	B	*1*	→	B	*0*	*0*	1	→	*1*	0	*1*	*2*	→	1	2	*2*	*2*	→	0
0	*0*	0	→	B	*1*	B	⋆	→	B	*0*	*0*	*1*	→	*1*	*0*	*1*	1	→	*1*	0	*2*	1	→	2
0	*0*	B	→	B	*0*	0	2	→	*0*	*0*	*0*	*2*	→	*0*	*0*	*1*	*1*	→	*1*	0	*2*	*2*	→	*0*
0	*0*	1	→	B	0	0	2	→	*0*	1	*0*	2	→	0	1	*1*	*1*	→	2	*0*	*2*	1	→	2
0	*0*	1	→	B	*0*	0	2	→	0	*1*	*0*	2	→	0	*1*	*1*	2	→	*2*	*0*	*2*	*2*	→	*0*
0	*0*	*1*	→	B	1	0	2	→	0	*0*	1	B	→	1	*1*	*1*	*2*	→	*2*	1	*2*	2	→	2
0	*0*	2	→	B	*1*	0	2	→	0	*0*	1	B	→	*0*	*0*	*1*	1	→	*1*	*1*	*2*	2	→	*2*
B	*0*	B	→	B	2	0	2	→	*0*	*0*	1	0	→	*0*	*0*	*1*	*1*	→	*1*	*1*	*2*	*2*	→	*2*
B	*0*	B	→	1	2	0	*2*	→	0	B	1	B	→	*0*	B	*1*	1	→	1	2	*2*	*2*	→	0
B	*0*	1	→	*1*	*0*	B	*0*	→	*0*	B	1	1	→	*0*	B	*1*	2	→	1	—	—	—	→	—
B	*0*	*1*	→	*1*	*0*	B	B	→	*0*	B	1	*1*	→	1	B	*1*	*2*	→	1					

(b) Symmetric rules

Figure 2.9: Transition table of $\mathfrak{F}'^{18}_{373}$ which uses `lvl` mod 2



(a) $\mathfrak{F}'^{18}_{373}$ with `lvl` mod 2 (b) $\mathfrak{F}'^{26}_{555}$ with `lvl` mod 3 (c) Umeo's solution $\mathfrak{U}^{6}_{78}$

Figure 2.10: Space-time diagrams for the FSSP 3n initial configuration of size 15

2.3.2 Improving the Finite CA $\mathfrak{F}^{21}_{486}$ by Quotients

We now turn to the goal of optimizing a given finite CA. We use the original finite-state field-based CA of Maignan and Yunès for historical reasons.

Background on Deterministic Finite Automata Minimization

When talking about automata optimization, one immediately thinks about the minimization of *Deterministic Finite Automata* (DFA for short) and there are a number of well-known algorithms like the Moore algorithm, the Brzozowski algorithm and the Hopcroft algorithm. A DFA receives an input word $u_1 u_2 \dots u_n$ build from its input alphabet, and transitions through a sequence $q_0 q_1 q_2 \dots q_n$ of states according to a transition function. Each state produces a bit of information called "accepting" or "rejecting". The collection gathering for each possible input word its last outputted bit of information is taken as a complete specification of the input-output behavior of the DFA and is usually formalized by the notion of recognized language. To minimize a DFA means to merge together its states in a coherent way so that the input-output behavior stays unchanged. It is known that starting with any DFA recognizing a language, such a merging of states produces the best possible DFA that recognizes this language.

For CA, things are more intricated and such a strong minimization is impossible but we can start by approaching the problem in a similar way. For the specific case

of FSSP, the input-output behavior is mainly specified by the fact the quiescent state should act as an inactive state and, considering the firing state as the only accepting state, by the fact that no accepting state should appears before the transition $2n-2$ and all states should be accepting at transition $t_s = 2n - 2$ for any length n.

Brute Force Exploration of All Quotients

As in the case of DFA minimization, we want to merge as many states of $\mathfrak{F}_{486}^{21}$ as possible while preserving the fact that it is a minimal-time FSSP solution. The merging of many states can always be obtained by merging two states, then another two states, and so on so forth. Also, merging two states might be described simply as a substitution of one of them by the other. However, the resulting object might not be a deterministic CA. Indeed, we might have a cellular automaton α with two transitions $(a, b, c) \mapsto d, (e, f, g) \mapsto h \in \delta_\alpha$ with $d \neq h$ and the substitution renders (a, b, c) equals to (e, f, g) but keeps $d \neq h$. In this case, the transition table is not a partial function anymore and we have a non-deterministic CA. However, if we then substitute d by h, and so on so forth every time we obtain a non-deterministic CA, we will necessarily end up with a deterministic CA at some point.

More precisely, let α be a CA, $e_0 \in \Sigma_\alpha$ be a state and e_1 an arbitrary element, a *substitution* of e_0 by e_1 in α gives a new (maybe non-deterministic) CA $\beta = \rho(e_0, e_1, \alpha)$ with $\Sigma_\beta = (\Sigma_\alpha \backslash \{e_0\}) \cup \{e_1\}$, $\mathrm{I}_\beta = \{c' \mid c \in \mathrm{I}_\alpha\}$ and $\delta_\beta = \{(a', b', c') \mapsto d' \mid (a, b, c) \mapsto d \in \delta_\alpha\}$ where

$$x' = \begin{cases} e_1 & \text{if } x = e_0 \\ x & \textit{otherwise.} \end{cases}$$

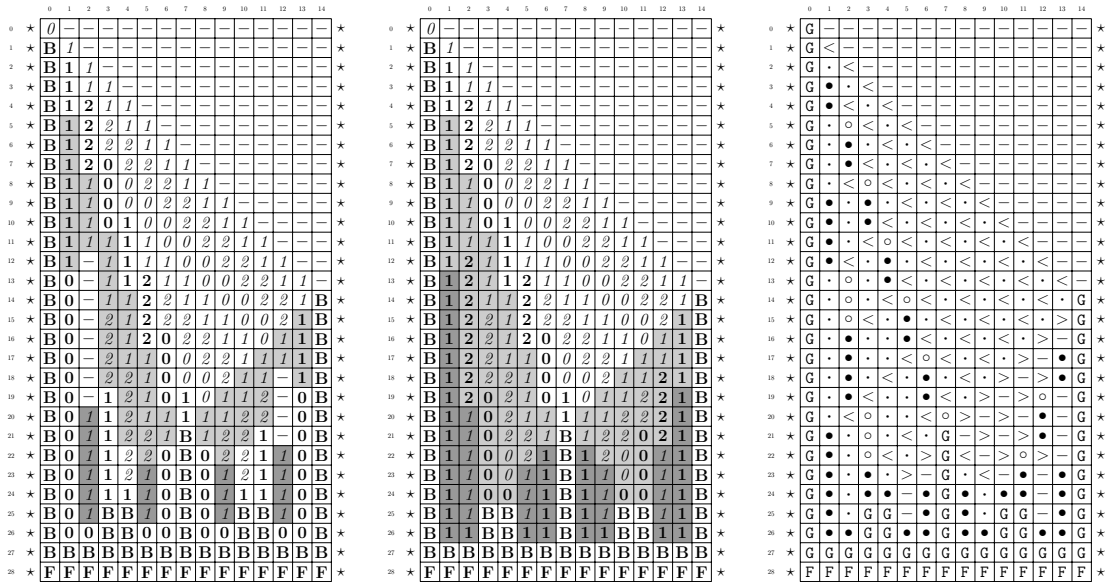
As we are considering FSSP solutions, we also need to keep track of the four special states and have $\star_\beta = \star_\alpha{}'$, $\mathtt{G}_\beta = \mathtt{G}_\alpha{}'$, $\mathtt{Q}_\beta = \mathtt{Q}_\alpha{}'$ and $\mathtt{F}_\beta = \mathtt{F}_\alpha{}'$.

We also define $\gamma = \rho^+(e_0, e_1, \alpha)$ the closest deterministic CA obtained by first computing $\beta = \rho(e_0, e_1, \alpha)$, and then taking $\gamma = \beta$ if β is deterministic. If β is not deterministic, then there exists $a, b, c, e_2, e_3 \in \Sigma_\beta^\star$ such that $(a, b, c) \mapsto e_2 \in \delta_\beta$ and $(a, b, c) \mapsto e_3 \in \delta_\beta$ but $e_2 \neq e_3$. In this case, we recursively set $\gamma = \rho^+(e_2, e_3, \beta)$. This operation is well defined up to a state renaming. We use the following straightforward algorithm to explore all quotients by brute force. This algorithm is indeed exhaustive because once a CA is not an FSSP solution, none of its quotients can be. Indeed, if it is not a FSSP solution, it is necessarily because the firing state occurs too early, and more merging can only make the firing states occur in more places in the space-time diagrams.

A Brief Analysis of The Quotients

We have implemented this algorithm in Java with an Ubuntu machine which had 2.70GHz processor speed. An execution of this algorithm on the CA $\mathfrak{F}_{486}^{21}$ does not produce a combinatorial explosion and stops after a few minutes. We found 3214 quotients, distinct even up to renaming: 30 quotients appear at recursion depth 1, 287 quotients at depth 2, 1041 quotients of depth 3, 1334 quotients of depth 4, 471 quotients of depth 5, 50 quotients of depth 6, and 1 reduction of depth 7. We verified that the 3214 CA are minimal-time FSSP solution by checking their space-time diagram for all FSSP initial configuration of size 2 to 1000. The following

```
AllQuotients(α)
 2 res := {}
 3 for {e1, e2} ⊆ Σ_α with e1 ≠ e2 do
 4 |   β := ρ+(e1, e2, α)
 5 |   if β is an FSSP solution then
 6 |   |   res := res ∪ {β}
 7 |   |   res := res ∪ AllQuotients(β)
 8 |   end
 9 end
10 return res
```



(a) reduction of $\mathfrak{F}^{14}_{480}$ by s_1 (b) Field Based CA $\mathfrak{F}^{21}_{486}$ (c) Noguchi's CA $\mathfrak{N}^{8}_{119}$

Figure 2.11: Space-time diagrams for the FSSP initial configuration of size 15

table lists for each number of states and each number of rules, the number of FSSP solution found as a quotient of $\mathfrak{F}^{21}_{486}$.

#states \ #rules	474	479	480	481	482	483	484	485	486
14			1						
15	1	3	25	6	15				
16	7	8	114	49	167	45	81		
17	12	6	146	79	385	91	335	112	168
18	7	1	53	31	243	39	262	113	292
19	1		5	3	50	4	61	29	134
20					3		4	2	21

All the other quotients that have been generated and declared non-solution merges the firing state with some other state, and in fact merges almost all states together to become deterministic. Another interesting fact is that all of the valid quotients only merges states two by two. In other words, changing $\rho^+(e_1, e_2, \alpha)$ into $\rho(e_1, e_2, \alpha)$ at line 4 and changing the test *β is an FSSP solution* by the test *β is a deterministic*

CA at line 5 in the algorithm lead to the same set of quotients. Fig. 2.11a shows a space-time diagram of the CA quotient of 14-state. It has 480 rules and is obtained by merging the following set of substitutions: (2, 0),(0, 2),(0, 2),(1, 0),(2, −),(0, 1),(0, 1). Note that these pairs are all disjoints.

2.4 Summary: Reduction Formulas and Quotients

Reduction formulas and quotient are basically the same mathematical concept. Indeed, they take as input a state and returns the new state that should be used instead. We use here reductions formulas when working on an infinite CA and (finite) quotients when working with finite CA. Because the transition table of a finite CA is a finite combinatorial object, and because of the simplicity of quotient (simple renaming of states), it is possible to apply the quotient directly on it the transition tables, and to explore quotient algorithmically as shown. But the results are very limited. Even if the 14-state *symmetric* FSSP solution, and should therefore be compared with other *symmetric* solutions to be fair, it is however the case that there are hand-made *non-symmetric* solutions with smaller number of state and we would like to be able to compare $\mathfrak{F}^{21}_{486}$ with them. For instance, look at Noguchi's 8-states solution depicted in Figure 2.11c. Of course, one can see that the quotient of a symmetric CA is necessarily a symmetric CA. Similarly, although the solution $\mathfrak{F}'^{26}_{555}$ (Figure 2.10b) has been designed to mimic the algorithm of Umeo's solution (Figure 2.10c), the latter can not be obtained as a quotient of the former, as we will see more precisely later on. There is thus a need to generalize the concept of quotient.

Chapter 3

From Quotients to Local Simulations

In this chapter, we generalize the notion of quotients to the notion of local simulations. This is done semi-formally to introduce the concept gently and match also more closely to the structure of the publication [27]. The precise formal definition is given in the next chapter.

3.1 Background: Noguchi's CAs $\mathfrak{N}^8_{119}$, $\mathfrak{N}^8_{134}$, $\mathfrak{N}^9_{141}$

Kenichiro Noguchi proposed two 8-state solutions and one 9-state solution for the FSSP as described in [29]. Between the two 8-state solutions, one solution have 134 rules and the other is an optimization to 119 rules that we denote $\mathfrak{N}^8_{134}$ and $\mathfrak{N}^8_{119}$ respectively. The 9-state solution is denoted $\mathfrak{N}^9_{141}$ and has 141 rules. The space-time diagrams of these solutions have the same structure as those of the field-based solution. However we obtained no 8-state solutions or 9-state solutions by the method described above. In order to study the relation between $\mathfrak{F}^{21}_{486}$ and these CA, we mainly focus on $\mathfrak{N}^8_{119}$, only providing some quick remarks on the other two CA.

Fig. 3.1 shows the transition table $\delta_{\mathfrak{N}^8_{119}}$. In each grid, the first line presents the current state e_0, the second line presents the state of the right neighbour e_1, the first column presents the state of the left neighbour e_{-1}. Each other cell of the table shows the results $\delta_{\mathfrak{N}^8_{119}}(e_{-1}, e_0, e_1)$. An empty cell means that there is no local transitions for (e_{-1}, e_0, e_1). Fig. 3.4a shows the space-time diagram of this CA on an FSSP initial configuration of size 15. In Fig. 3.1 and Fig. 3.4a, $\boxed{\mathtt{G}}$ denotes the general $\mathtt{G}_{\mathfrak{N}^8_{119}}$, $\boxed{-}$ the quiescent state $\mathtt{Q}_{\mathfrak{N}^8_{119}}$ and $\boxed{\mathtt{F}}$ denotes the firing state $\mathtt{F}_{\mathfrak{N}^8_{119}}$. Note that this solution is "quasi-symmetric". In the active part of this space-time diagram, one can see that any state of $\mathfrak{N}^8_{119}$ has an associated symmetric state *i.e.* two states contain the same information but in the opposite direction. The state $\boxed{<}$ is reversed into $\boxed{>}$, the state $\boxed{\cdot}$ into $\boxed{-}$, and the four states $\boxed{\mathtt{G}}$, $\boxed{\mathtt{F}}$, $\boxed{\circ}$, and $\boxed{\bullet}$ are self-symmetric.

Note also that the local transitions table of CA $\mathfrak{N}^8_{134}$ was not given in [29] but we have been able to reconstruct it. Indeed, the space-time diagram of size 22 of this solution was given. This space-time diagram allowed to retrieved 114 local transitions. To find the other 20 local transitions, 18 local transitions was taken from the original CA $\mathfrak{N}^8_{119}$. The two other missing local transitions are obvious to deduce. Figures 3.2 and 3.3 shows the transition tables of the two others Noguchi's solutions and space-time diagrams of the three solutions are depicted in Figure 3.4.

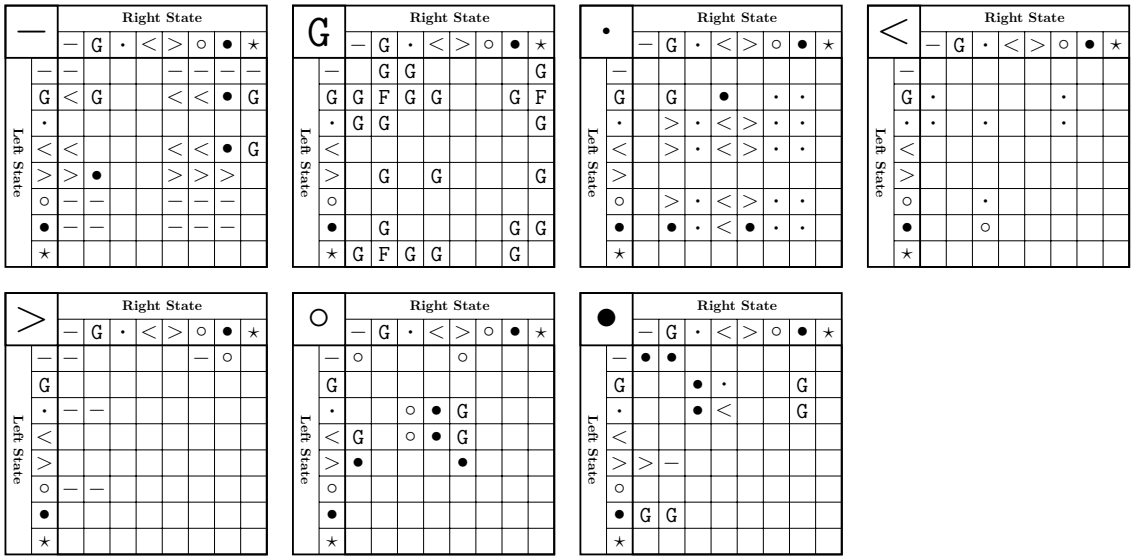


Figure 3.1: Transition table of an 8-state and 119-rule Noguchi's CA $\mathfrak{N}^8_{119}$

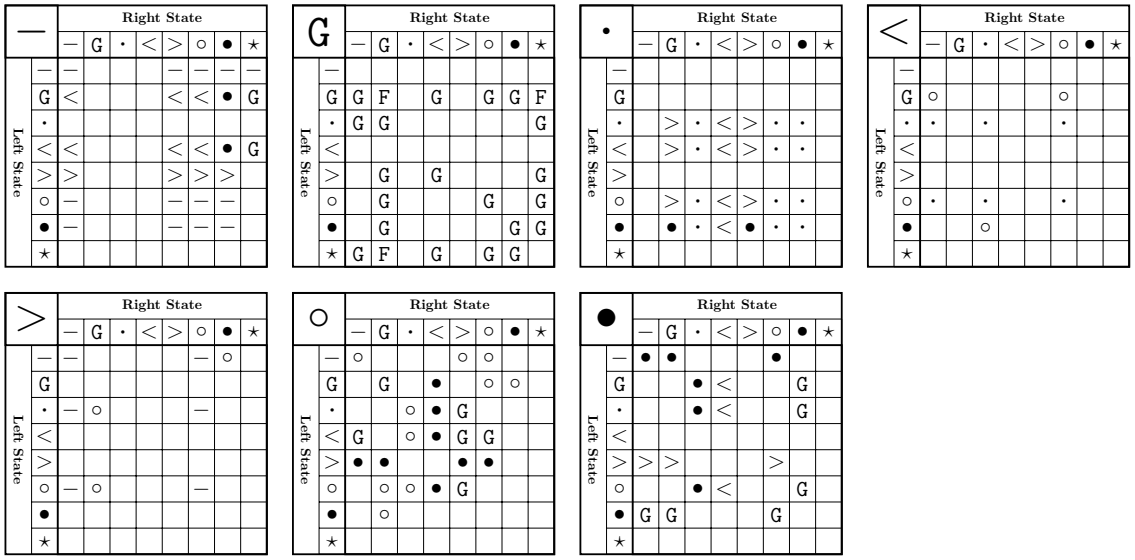


Figure 3.2: Transition table of an 8-state and 134-rule Noguchi's CA $\mathfrak{N}^8_{134}$

3.2 Comparison of $\mathfrak{F}^{21}_{486}$ with Noguchi's solutions

3.2.1 Local Simulation of $\mathfrak{F}^{21}_{486}$ into $\mathfrak{N}^8_{119}$

Although the space-time diagram of $\mathfrak{N}^8_{119}$ looks similar to the one of $\mathfrak{F}^{21}_{486}$, $\mathfrak{N}^8_{119}$ is not a quotient of $\mathfrak{F}^{21}_{486}$ since we did not obtained any 8-state quotient in the exhaustive exploration of the previous chapter. It is expected since that transition table of $\mathfrak{N}^8_{119}$ is non symmetric, i.e. there are local configurations that give different results when reversed. However, looking only a local part of the space-time diagram of $\mathfrak{F}^{21}_{486}$ seemed enough to determine the corresponding local part of the space-time diagram of $\mathfrak{N}^8_{119}$. More precisely, one can check on Fig. 3.5 that at any time t and any position p, the local configuration $(\mathrm{D}_{\mathfrak{F}^{21}_{486}}(\overline{15})(t,p-1), \mathrm{D}_{\mathfrak{F}^{21}_{486}}(\overline{15})(t,p), \mathrm{D}_{\mathfrak{F}^{21}_{486}}(\overline{15})(t,p+1))$ on the space-time diagram of $\mathfrak{F}^{21}_{486}$ determines the state $\mathrm{D}_{\mathfrak{N}^8_{119}}(\overline{15})(t+1,p)$ on the space-time diagram of $\mathfrak{N}^8_{119}$, i.e. if this local configuration appears at some other place

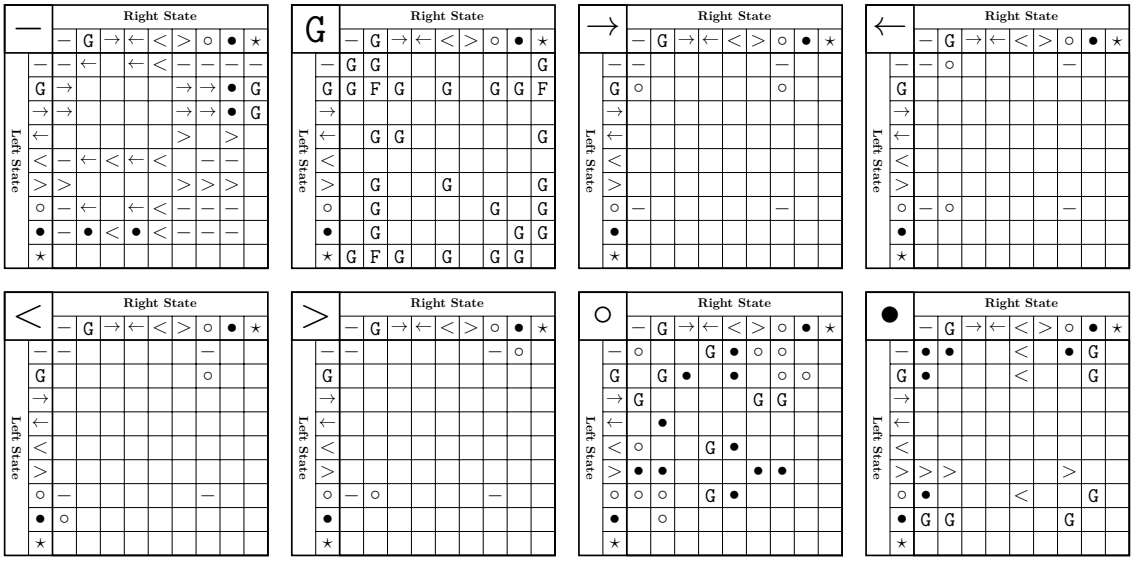


Figure 3.3: Transition table of an 9-state and 141-rule Noguchi's CA $\mathfrak{N}^{9}_{141}$

(t', p') on $\mathfrak{F}^{21}_{486}$, leads to the same states at $(t'+1, p')$ on $\mathfrak{N}^{8}_{119}$.

Formally, we consider a function $f : \Sigma^{\star}_{\mathfrak{F}^{21}_{486}} \times \Sigma_{\mathfrak{F}^{21}_{486}} \times \Sigma^{\star}_{\mathfrak{F}^{21}_{486}} \to \Sigma_{\mathfrak{N}^{8}_{119}}$ defined as :

$$f = \{(\mathrm{D}_{\mathfrak{F}^{21}_{486}}(\overline{n})(t, p-1), \mathrm{D}_{\mathfrak{F}^{21}_{486}}(\overline{n})(t, p), \mathrm{D}_{\mathfrak{F}^{21}_{486}}(\overline{n})(t, p+1)) \mapsto \mathrm{D}_{\mathfrak{N}^{8}_{119}}(\overline{n})(t+1, p)\}$$

for any $n \in \mathbb{N}^{+}, t \in \mathbb{N}$ and $p \in [\![1, n]\!]$. We call this function a *local mapping*. We approximated it by taking it with $n \in [\![2, 1000]\!]$, i.e. we only considered the FSSP initial configuration of size 2 to 1000, but the function stayed unchanged above $n = 105$. This is indeed the size at which all local transition of $\mathfrak{F}^{21}_{486}$ appears. The check until $n = 1000$ was still necessary to check informally that the function is indeed well defined, and is not a mere non-functional relation.

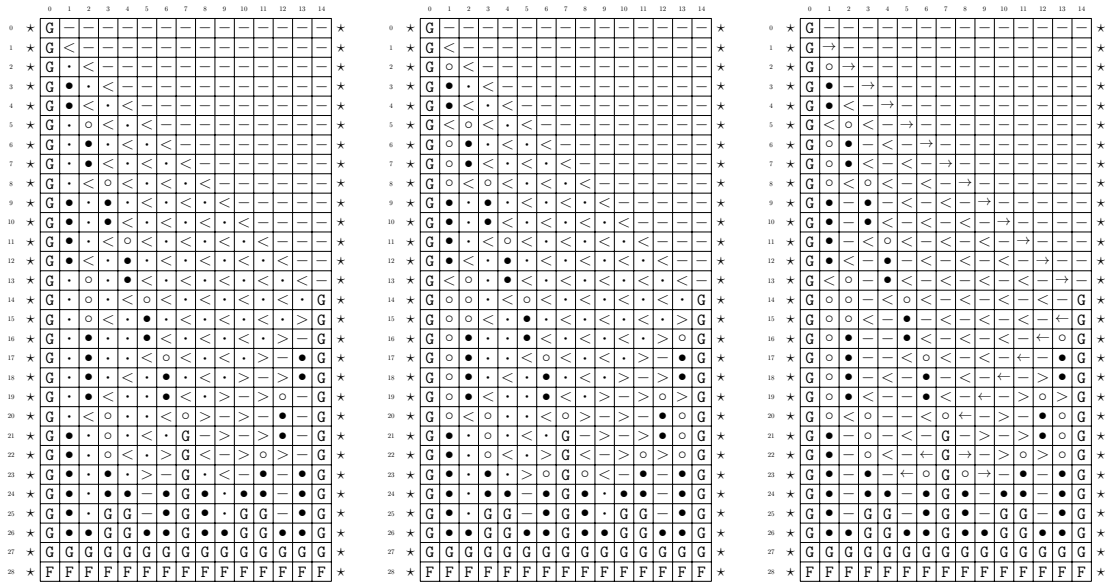
A Brief Analysis of The Result

We have built the local mapping which have the same number of elements with the local transition table $\delta_{\mathfrak{F}^{21}_{486}}$. Each element of f of the form $(a, b, c) \mapsto d$ with $a, c \in \Sigma^{\star}_{\mathfrak{F}^{21}_{486}}, b \in \Sigma_{\mathfrak{F}^{21}_{486}}, d \in \Sigma_{\mathfrak{N}^{8}_{119}}$ is indeed well-defined. This is indeed a notion of quotient because the CA $\mathfrak{N}^{8}_{119}$ can be recovered from the CA $\mathfrak{F}^{21}_{486}$ in the following way. From the CA $\mathfrak{F}^{21}_{486}$, we first consider its family of space-time diagram (up to a certain size). Using the local mapping, we can transform those space-time diagrams to obtain a new family of space-time diagram. This new family being deterministic, we can obtain $\mathfrak{N}^{8}_{119}$ as the CA associated to this deterministic family.

Moreover, this is a proper generalization because the previous notion of quotient can always be described as a particular case of this new notion. Indeed, a quotient from a CA α transform each state $e \in \Sigma_{\alpha}$ to some state $g(e)$. The associated local mapping f is simply defined as $f(a, b, c) = g(\delta_{\alpha}(a, b, c))$.

The reduction formulas from $\mathfrak{F}^{21}_{486}$ to $\mathfrak{N}^{8}_{119}$ and $\mathfrak{N}^{8}_{134}$

We want to explain the local mapping in a more intelligible way. To do so, we present the latter by formulas describing when and where the $\mathfrak{N}^{8}_{119}$ states appears in terms of the 6 data composing the $\mathfrak{F}^{21}_{486}$ states.



(a) Noguchi's CA $\mathfrak{N}^{8}_{119}$ (b) Noguchi's CA $\mathfrak{N}^{8}_{134}$ (c) Noguchi's CA $\mathfrak{N}^{9}_{141}$

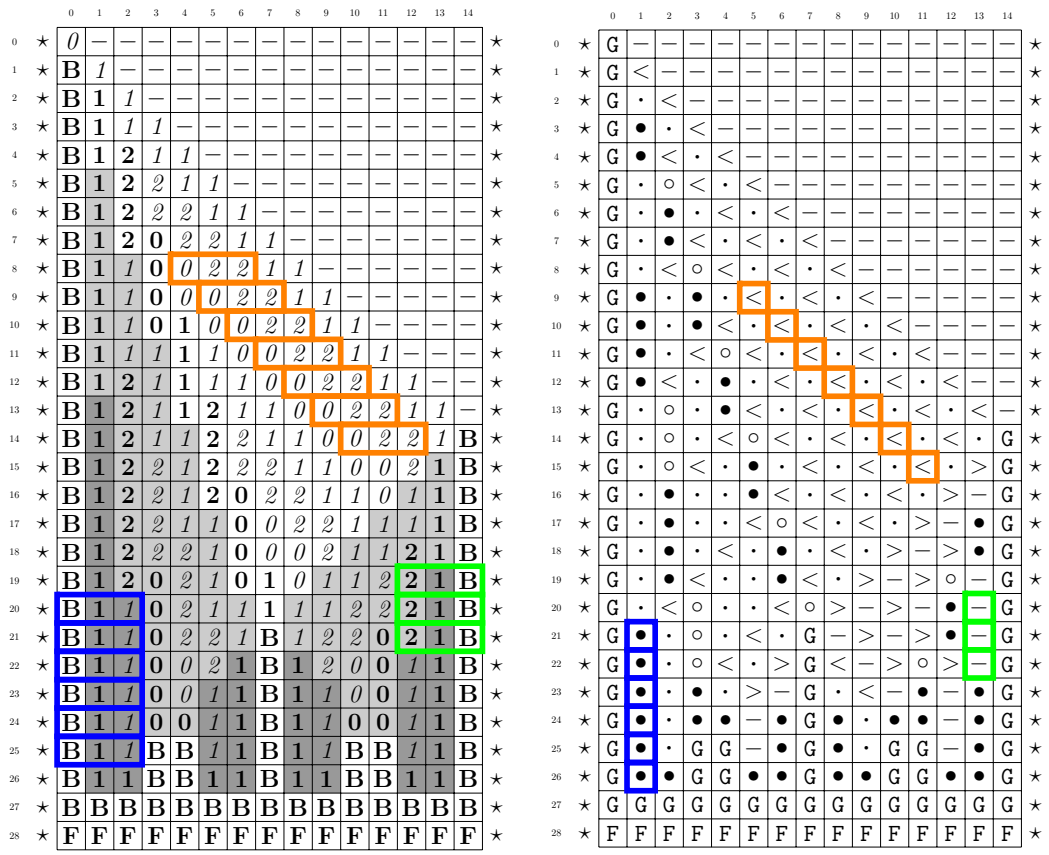
Figure 3.4: Space-time diagrams for the FSSP initial configuration of size 15

One important aspect of this reduction is that it goes from a symmetric solution $\mathfrak{F}^{21}_{486}$ to a quasi-symmetric solution $\mathfrak{N}^{8}_{119}$ as noted in Section 2.4 and Section 3.2 respectively. This means that all informations of $\mathfrak{N}^{8}_{119}$ comes with an orientation: either left or right. For example, two symmetric local configurations ($\boxed{\mathbf{2}}$, $\boxed{2}$, $\boxed{2}$) and ($\boxed{2}$, $\boxed{2}$, $\boxed{\mathbf{2}}$) of $\mathfrak{F}^{21}_{486}$ have same result $\boxed{\mathbf{0}}$. The local simulation of CA $\mathfrak{F}^{21}_{486}$ into CA $\mathfrak{N}^{8}_{119}$ gives ($\boxed{\mathbf{2}}$, $\boxed{2}$, $\boxed{2}$) $\mapsto \boxed{<}$ and ($\boxed{2}$, $\boxed{2}$, $\boxed{\mathbf{2}}$) $\mapsto \boxed{>}$. We begin be describing how this direction can be obtained from the $\mathfrak{F}^{21}_{486}$ states. We then express the $\mathfrak{N}^{8}_{119}$ states using the 6 data contained in the states of the $\mathfrak{F}^{21}_{486}$ and this new direction information.

The direction information of CA $\mathfrak{F}^{21}_{486}$ We denote the direction information by `dir`. It has 4 possible values : a special quiescent value $\boxed{-}$, no direction for border $\boxed{\|}$, a direction to the right $\boxed{\rightarrow}$, a direction to the left $\boxed{\leftarrow}$. For a local transition $(l, c, r) \mapsto n$ of CA $\mathfrak{F}^{21}_{486}$, where l, c, r, and n stands for "left", "center", "right", and "new" respectively, the direction information is calculated with the formula :

$$
\mathtt{dir}_n = \begin{cases}
\boxed{-} & \text{if } \ \neg \mathtt{inp}_n; \\
\boxed{\|} & \text{if } \ \mathtt{rbrd}_n; \\
\boxed{\rightarrow} & \text{if } \ \mathtt{inp}_n \wedge (\neg\, \mathtt{inp}_r \vee (\mathtt{rbrd}_l \wedge \neg\, \mathtt{rbrd}_r) \vee \\
 & \quad (\mathtt{rlvl}_l = \mathtt{rlvl}_n \wedge \neg\, \mathtt{rsta}_n \wedge \mathtt{rdst}_l = \mathtt{rdst}_n) \vee \\
 & \quad (\mathtt{rlvl}_l = \mathtt{rlvl}_n \wedge \neg\, \mathtt{rsta}_n \wedge \mathtt{rdst}_l = (\mathtt{rdst}_n + 1) \bmod 3) \vee \\
 & \quad (\mathtt{rlvl}_l = \mathtt{rlvl}_n \wedge \ \ \mathtt{rsta}_n \wedge \mathtt{rsta}_l \wedge (\mathtt{rdst}_l + 1) \bmod 3 = \mathtt{rdst}_n) \vee \\
 & \quad (\mathtt{rlvl}_l = (\mathtt{rlvl}_n + 1) \bmod 3)); \\
\boxed{\leftarrow} & \text{if } \ \mathtt{inp}_n \wedge (\neg\, \mathtt{inp}_l \vee (\mathtt{rbrd}_r \wedge \neg\, \mathtt{rbrd}_l) \vee \\
 & \quad (\mathtt{rlvl}_r = \mathtt{rlvl}_n \wedge \neg\, \mathtt{rsta}_n \wedge \mathtt{rdst}_r = \mathtt{rdst}_n) \vee \\
 & \quad (\mathtt{rlvl}_r = \mathtt{rlvl}_n \wedge \neg\, \mathtt{rsta}_n \wedge \mathtt{rdst}_r = (\mathtt{rdst}_n + 1) \bmod 3) \vee \\
 & \quad (\mathtt{rlvl}_r = \mathtt{rlvl}_n \wedge \ \ \mathtt{rsta}_n \wedge \mathtt{rsta}_r \wedge (\mathtt{rdst}_r + 1) \bmod 3 = \mathtt{rdst}_n) \vee \\
 & \quad (\mathtt{rlvl}_r = (\mathtt{rlvl}_n + 1) \bmod 3)).
\end{cases}
$$

Note that the two formulas of $\boxed{\rightarrow}$ and $\boxed{\leftarrow}$ are the same. As expected, passing from one to the other is a matter of reversing the left and right states in the two formulas.



(a) Field Based CA $\mathfrak{F}^{21}_{486}$ (b) Noguchi's CA $\mathfrak{N}^{8}_{119}$

Figure 3.5: Correspondence of local configurations of $\mathfrak{F}^{21}_{486}$ to states of $\mathfrak{N}^{8}_{119}$ in their space-time diagrams for the FSSP initial configuration of size 15.

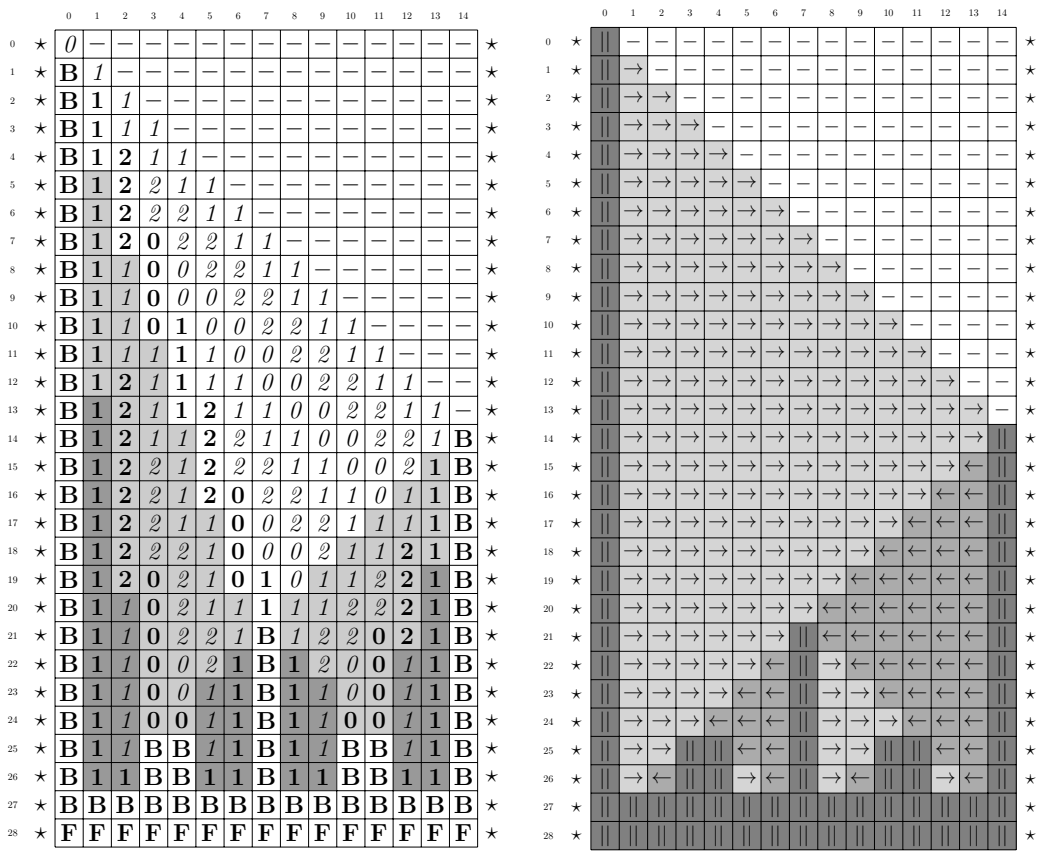
Fig. 3.6b shows the space-time diagram of the direction information with initial configuration of size 15. In the following, we use the variable b and f, for "back" and "front", to talk about the left or right state relatively to the direction, *i.e.* the local transition is $(b, c, f) \mapsto n$ when $\texttt{dir}_n = \boxed{\rightarrow}$ and $(f, c, b) \mapsto n$ when $\texttt{dir}_n = \boxed{\leftarrow}$.

The reduction formulas of CA $\mathfrak{F}^{21}_{486}$ into CA $\mathfrak{N}^{8}_{119}$ In the Fig. 2.11b and Fig. 2.11c, it is easy to see that the two states $\boxed{\texttt{G}}$ and $\boxed{\texttt{F}}$ of CA $\mathfrak{N}^{8}_{119}$ corresponds exactly to the two states $\boxed{\texttt{B}}$ and $\boxed{\texttt{F}}$ of CA $\mathfrak{F}^{21}_{486}$ (except for the initial configuration). We explain this correspondence with the two simple formulas:

$$\boxed{\texttt{G}} = \texttt{rbrd}_n, \text{ and}$$
$$\boxed{\texttt{F}} = \texttt{rout}_n.$$

In the active area, one can see that the two states $\boxed{<}$ and $\boxed{>}$ appear in $\mathfrak{N}^{8}_{119}$ when, in $\mathfrak{F}^{21}_{486}$, a cell sees its `rdst` value change but not its level, or when `rlvl` changes and `rsta` is false at current or previous timestep. All these cases occur for the seventh cell from the left in Fig 2.11b and 2.11c. Formally:

$$\boxed{>} = \bigvee \begin{cases} \texttt{dir}_n = \boxed{\leftarrow} \wedge \texttt{rlvl}_n = \texttt{rlvl}_c \wedge \texttt{rdst}_n \neq \texttt{rdst}_c, \\ \texttt{dir}_n = \boxed{\leftarrow} \wedge \texttt{rlvl}_n \neq \texttt{rlvl}_c \wedge (\neg\, \texttt{rsta}_c \vee \neg\, \texttt{rsta}_n), \text{ and} \end{cases}$$
$$\boxed{<} = \bigvee \begin{cases} \texttt{dir}_n = \boxed{\rightarrow} \wedge \texttt{rlvl}_n = \texttt{rlvl}_c \wedge \texttt{rdst}_n \neq \texttt{rdst}_c, \\ \texttt{dir}_n = \boxed{\rightarrow} \wedge \texttt{rlvl}_n \neq \texttt{rlvl}_c \wedge (\neg\, \texttt{rsta}_c \vee \neg\, \texttt{rsta}_n). \end{cases}$$



(a) Field Based CA $\mathfrak{F}^{21}_{486}$ (b) Field direction

Figure 3.6: Space-time diagrams of $\mathfrak{F}^{21}_{486}$ and a field direction initial configuration of size 15

The main feature of the states $\boxdot$, $\boxminus$, $\boxbullet$, and $\boxcirc$ can be observed with the third cell in the same figures. Indeed, from time 9 to 11, this cell has state $\boxdot$ in $\mathfrak{N}^8_{119}$ when there is no state change in $\mathfrak{F}^{21}_{486}$. At time 12, its state is $\boxleq$ in $\mathfrak{N}^8_{119}$ because of the change of `rdst` value in $\mathfrak{F}^{21}_{486}$. After that, there is a sequence of $\boxcirc$ followed by a sequence of $\boxbullet$ in $\mathfrak{N}^8_{119}$ while the state does not change at all in $\mathfrak{F}^{21}_{486}$. However, this change of state in $\mathfrak{N}^8_{119}$ corresponds exactly to the time at which the cell observes a change of distance value in its right neighbor. Of course, the same thing happens in the other direction, *e.g.* cell 13, but with the state $\boxdot$ changed into $\boxminus$ and the left instead of the right neighbor. This is the main content of the following reduction formulas. We also need to take into account that state $\boxminus$ also represents the quiescent state, and some particularities of $\mathfrak{N}^8_{119}$ which appears because of the optimizations of Noguchi on which we comment in the following section.

$$\boxminus = \bigvee \begin{cases} \neg\, \mathtt{inp}_n, \\ \mathtt{dir}_n = \boxleftarrow \wedge \neg\, \mathtt{rsta}_n \wedge n = c, \\ \mathtt{dir}_n = \boxleftarrow \wedge \mathtt{rlvl}_n = (\mathtt{rlvl}_f + 1) \bmod 3 \wedge \mathtt{dir}_n = \mathtt{dir}_c, \text{ and} \end{cases}$$

$$\boxdot = \bigvee \begin{cases} \mathtt{dir}_n = \boxrightarrow \wedge \mathtt{rsta}_n \wedge \neg\, \mathtt{inp}_f, \\ \mathtt{dir}_n = \boxrightarrow \wedge \neg\, \mathtt{rsta}_n \wedge n = c, \\ \mathtt{dir}_n = \boxrightarrow \wedge \mathtt{rlvl}_n = (\mathtt{rlvl}_f + 1) \bmod 3 \wedge \mathtt{dir}_n = \mathtt{dir}_c. \end{cases}$$

$$\boxcirc = \mathtt{dir}_n \in \{\boxrightarrow, \boxleftarrow\} \wedge \mathtt{rsta}_n \wedge n = c \wedge \mathtt{rdst}_n = (\mathtt{rdst}_f + 1) \bmod 3.$$

$$\boxbullet = \mathtt{dir}_n \in \{\boxrightarrow, \boxleftarrow\} \wedge \mathtt{rsta}_n \wedge \mathtt{rdst}_n = \mathtt{rdst}_c \wedge \mathtt{rdst}_n = \mathtt{rdst}_f.$$

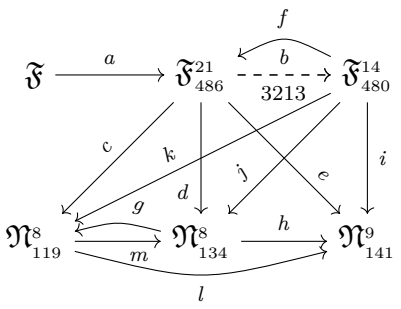


Figure 3.7: Relation between the solutions at study in this paper.

The reduction formulas of CA $\mathfrak{F}^{21}_{486}$ into CA $\mathfrak{N}^{8}_{134}$ In the same way, we have found the reduction formulas for recognizing the CA $\mathfrak{N}^{8}_{134}$ from the CA $\mathfrak{F}^{21}_{486}$. The reduction formulas of states G, F, • are the sames for CA $\mathfrak{N}^{8}_{119}$ and $\mathfrak{N}^{8}_{134}$. Note that the reduction formulas for four states −, ·, >, and < are more simple in $\mathfrak{N}^{8}_{134}$ than in $\mathfrak{N}^{8}_{119}$. This is because this version of the CA is not optimized, so that each state has a much clearer responsibility. For the five states ·, −, >, < and ○ of CA $\mathfrak{N}^{8}_{134}$, the reduction formulas are given as:

$$
\begin{aligned}
\cdot &= \ \mathtt{dir}_n = \rightarrow \wedge \neg\, \mathtt{rsta}_n \wedge n = c,\\
- &= (\mathtt{dir}_n = \leftarrow \wedge \neg\, \mathtt{rsta}_n \wedge n = c) \vee \neg\, \mathtt{inp}_n,\\
> &= \mathtt{dir}_n = \leftarrow \wedge (\mathtt{rdst}_n \neq \mathtt{rdst}_c \vee \mathtt{rlvl}_n \neq \mathtt{rlvl}_c),\\
< &= \mathtt{dir}_n = \rightarrow \wedge (\mathtt{rdst}_n \neq \mathtt{rdst}_c \vee \mathtt{rlvl}_n \neq \mathtt{rlvl}_c), \text{ and}\\
\circ &= \mathtt{dir}_n \in \{\rightarrow, \leftarrow\} \wedge \mathtt{rsta}_n \wedge \mathtt{rdst}_n = \mathtt{rdst}_c \wedge \mathtt{rlvl}_n = \mathtt{rlvl}_c\\
&\qquad \wedge (\mathtt{rdst}_n = (\mathtt{rdst}_f + 1) \bmod 3 \vee \mathtt{rlvl}_n = (\mathtt{rlvl}_f + 1) \bmod 3).
\end{aligned}
$$

3.2.2 Analyzing the relations between the solutions

Fig. 3.7 shows the relations between the solutions studied in this chapter. An arrow goes from CA α to CA β when CA β is a reduction of CA α. The arrow a is the first reduction described in [18]. In this paper, with the reduction by substitution, we have build a collection of arrows b by passing through 3213 intermediate CA. The arrows c, d, e are the local simulation that we talked about in Section 3.2. We also use the local simulation to check the relation between other solutions and we found the following:

- The three CA of Noguchi are also reductions of $\mathfrak{F}^{14}_{480}$ (i, j and k).
- The two CA $\mathfrak{N}^{8}_{119}$ and $\mathfrak{N}^{8}_{134}$ are reductions of each other (g and m).
- Some local simulations increase the number of states (f, h and l).
- We have no arrow from the Noguchi's solutions into the field-based solutions.

3.3 Comparison of Clergue et al.'s 718 solutions

The former results shows that the field-based approach should be able to generate an 8-state solution from the high-level description. One can ask whether it is possible

to generate a 6-state solution. Let us come back on the 718 solutions found in [3] and mentioned in Chapter 1. Look from a local mapping from $\mathfrak{F}_{486}^{21}$ or from $\mathfrak{F}_{329}^{15}$ to these 718 solutions does not lead to any result unfortunately. This does not means that the infinite cellular automaton $\mathfrak{F}$ cannot be reduced into a 6-state solution, but at this stage, a interesting question arises naturally.

Indeed, the notion of local simulation can be though of as a new way of grouping CA together. Looking at the 718 solutions, a natural idea is that many solutions are really just one solution slightly modified. Can the notion of local mapping help to organize these solutions in equivalence classes exhibiting which of these solutions have, informally speaking, the "same algorithm". These solutions, numbered from 0 to 717, are freely available online, so we therefore tried to search local simulations between them. Firstly, we found a slight mistake since there are 12 pairs of equivalent solutions up to renaming of states: (105, 676), (127, 659), (243, 599), (562, 626), (588, 619), (601, 609), (603, 689), (611, 651), (629, 714), (663, 684), (590, 596) and (679, 707). This means that there are really 706 solutions, but we still refer to them as the 718 solutions with their original numbering.

Once local simulations established between the 718 solutions, we analyzed the number of connected components and found 193 while expecting only a few. We expected ideally 2 connected components in fact: one for the halving-based solutions, and one for the Mazoyer-like solutions. At this point, many things need to be said to advance more on this subject, but this is not the right places to engage in further new concepts as this might make the next chapter harder to follow. So these concepts will be discussed in the final chapter.

3.4 Conclusion

We found a 14-states quotients of a field-based solution to the FSSP $\mathfrak{F}_{486}^{21}$. Note that the original solution is designed for the generalized FSSP where the general can be at any position. We also have shown in which sense the two Noguchi's solutions can be viewed as a particular reduction of the field-based solution via the notion of local mapping and local simulation. We have explained these reductions with the formulas. Some local simulations increase the number of states. This is expected since we first virtually augment the "number of states" by passing to the local configurations and then reduce from them. This notion is therefore not about decreasing the number of states but about changing the local encoding of the information while keeping the "algorithm" identical. Finally we studied how the 718 6-state solutions of Clerge's et al. can be grouped into groups of solutions differing only by their local encoding of information (in this precise sense) and showed that there are thus essentially 193 solutions only at this level of grouping.

Chapter 4

Exploiting Local Simulations

With the concept of local simulation, we have been able to demonstrate the relation between several FSSP solutions, and more particularly between solutions with a given number of states and transitions and solutions with fewer states or fewer transitions. However, the goal is to *generate* solutions with fewer states or fewer transitions. In this quest, we have tried different approaches (*e.g.*, genetic algorithm), but finally one of them produced so many solutions that a detour has been taken to have a better grasp on what was happening and to make sure that there were indeed millions of solutions even with just six states, and even after discarding permutation-equivalent ones.

In the course of the generation, different kinds of intermediary objects are manipulated, which might lack some of the data or some of the properties we want at the end. We begin by defining these objects in section 4.1. In Section 4.2, we present nice properties relating these objects and allowing the search algorithm to save a huge amount of time. Without these properties, it would have not been practically tractable to generated all these solutions. In Section 4.3, we describe the exploration algorithm and continue in Section 4.4 with some experimental results: millions of solution with little effort, making think that there might be billions of solution eventually. We conclude in Section 5.4 with some formal and experimental future work.

4.1 Formalizing Local Simulations

In this section, we define formally local mappings and FSSP-candidate CAs in a way suitable to the current study. The material here is a considerable re-organization of the material found in [27].

4.1.1 FSSP-candidate

Definition 9. *A cellular automaton α is* FSSP-candidate *if there are four special states $\star_\alpha, \mathtt{G}_\alpha, \mathtt{Q}_\alpha, \mathtt{F}_\alpha \in \Sigma_\alpha$ satisfying the following conditions:*

1. *$\star_\alpha$ is an* outside state*,* i.e. *$\forall\, l \in dom(\delta_\alpha)$, $\delta_\alpha(l) = \star_\alpha \Leftrightarrow l(0) = \star_\alpha$.*
2. *$\mathtt{G}_\alpha$ is a* general state*,* i.e. *$\mathrm{I}_\alpha = \{\overline{n}_\alpha \mid n \geq 2\}$ with $\overline{n}_\alpha \in {\Sigma_\alpha}^{\mathbb{Z}}$ being the* FSSP initial configuration of size n *defined as $\overline{n}_\alpha(p) = \star_\alpha$, $\mathtt{G}_\alpha$, $\mathtt{Q}_\alpha$, $\star_\alpha$ for $p < 0$, $p = 0$, $0 < p < n$, and $p \geq n$ respectively.*

3. $\mathtt{Q}_\alpha$ *is a* quiescent state, i.e. $\delta_\alpha(\mathtt{Q}_\alpha, \mathtt{Q}_\alpha, \mathtt{Q}_\alpha) = \delta_\alpha(\mathtt{Q}_\alpha, \mathtt{Q}_\alpha, \star_\alpha) = \mathtt{Q}_\alpha$.

In other words, a FSSP-candidate CA is almost a FSSP solution as in Def 4, except that it does not necessarily have the synchronization condition (4) of this definition.

4.1.2 Local Mappings and Local Simulations

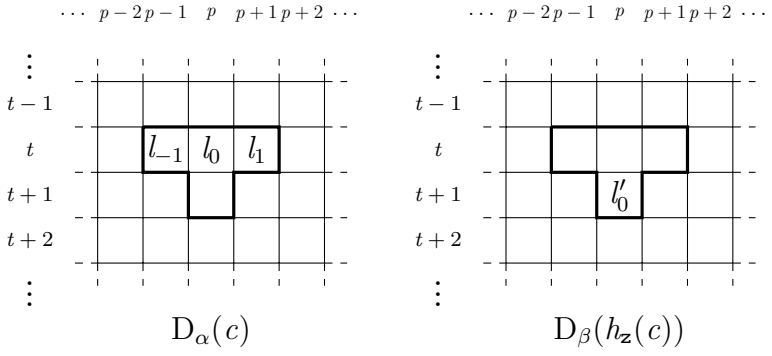
These two concepts are more easily pictured with space-time diagrams. Given a space-time diagram $d \in S^{\mathbb{N}\times\mathbb{Z}}$, we build a new one d' where each state $d'(t,p)$ is computed through a function h on the little cone $\langle d(t-dt, p+dp) \mid dt \in \{0,1\}, dp \in [\![-dt, +dt]\!]\rangle$ in d. This cone is simply a state for $t = 0$, and when d is generated by a CA, this cone is entirely determined by $\langle d(t-1, p+dp) \mid dp \in [\![-1,1]\!]\rangle$ for $t \geq 1$. Since the set of all these triplets is exactly $\mathrm{dom}(\delta_\alpha)$ (the triplets of the transition table), the following definitions suffice for the current study. We call this h a local mapping, because the new diagram $d' = h(d)$ is determined locally by the original one. When transforming a deterministic family, the result might not be deterministic, but if it is, we speak of a local simulation between two CA.

Definition 10. *A* local mapping ℓ from a CA α to a finite set X *consists of two functions* $\ell_{\mathtt{z}} : \{d(0,p) \mid (d,p) \in \mathrm{D}_\alpha \times \mathbb{Z}\} \to X$ *and* $\ell_{\mathtt{s}} : dom(\delta_\alpha) \to X$. *We define its* associated family of diagrams $\Phi_\ell = \{\ell(d) \mid d \in \mathrm{D}_\alpha\}$ *where:*

$$\ell(d)(t,p) = \begin{cases} \ell_{\mathtt{z}}(d(0,p)) & \text{if } t = 0, \\ \ell_{\mathtt{s}}(d(t-1,p-1), d(t-1,p), d(t-1,p+1)) & \text{if } t > 0. \end{cases}$$

If Φ_ℓ *is deterministic, we say that* ℓ *is a* local simulation from CA α to CA Γ_{Φ_ℓ}.

Proposition 1. *Equivalently, a* local simulation h from a CA α to a CA β *is a local mapping from* α *to the set* Σ_β *such that* $\{h_{\mathtt{z}}(c) \mid c \in \mathrm{I}_\alpha\} = \mathrm{I}_\beta$ *and for all* $(c,t,p) \in \mathrm{I}_\alpha \times \mathbb{N} \times \mathbb{Z}$, *we have* $h_{\mathtt{s}}(l_{-1}, l_0, l_1) = l'_0$ *with* $l_i = \mathrm{D}_\alpha(c)(t, p+i)$ *and* $l'_0 = \mathrm{D}_\beta(h_{\mathtt{z}}(c))(t+1, p)$. *The details of these formula are more easily seen graphically.*



Proof

The first direction of this equivalence means that if we have a local mapping h from α to X such that Φ_h is deterministic, if we also have $\beta = \Gamma_{\Phi_h}$, then it follows that $X = \Sigma_\beta$, $\{h_{\mathtt{z}}(c) \mid c \in \mathrm{I}_\alpha\} = \mathrm{I}_\beta$ and for all $(c, t, p) \in \mathrm{I}_\alpha \times \mathbb{N} \times \mathbb{Z}$, we have $h_{\mathtt{s}}(l_{-1}, l_0, l_1) = l'_0$ with $l_i = \mathrm{D}_\alpha(c)(t, p+i)$ and $l'_0 = \mathrm{D}_\beta(h_{\mathtt{z}}(c))(t+1, p)$.
Indeed, by Def 7 of Γ, having $\beta = \Gamma_{\Phi_h}$ means that $\Sigma_\beta = X$, $\{\mathrm{D}_\beta(c) \mid c \in \mathrm{I}_\beta\} = \Phi_h$. So the diagrams of β and the diagrams generated by h are identical, and the claimed properties are just a rewriting of the defining properties of Φ_h given in Def 10.
The other direction is a similar rewriting.

In other words, the definition focuses on the generation of Γ_{Φ_ℓ} from α while the proposition focuses on the situation where α and β are given and looking from the local simulation relationship as in this previous chapter.

4.2 Some Useful Algorithmic Properties

Our global strategy to find new FSSP solutions is to build them from local simulations of already existing FSSP solution α. Taking literally the previous definitions could lead to the following procedure for a given local mapping h. First, generate sufficiently many space-time diagrams of D_α. Secondly, use h to transform each of these diagrams $d \in \mathrm{D}_\alpha$ into a new one $h(d)$, thus producing a sub-family of Φ_h. At the same time, build the local transition relation δ_{Φ_h} of Φ_h by collecting all local transitions appearing in each $h(d)$ and check firsty for determinism and secondly for correct synchronization. If everything goes fine, we have a new FSSP solution $\beta = \Gamma_{\Phi_h}$.

Such a procedure is time-consuming. We show here useful properties that reduces drastically this procedure to a few steps. In fact, the space-time diagrams of Φ_h never need to be computed, neither to build the local transition relation δ_{Φ_h} (Section 4.2.1), nor to check correct synchronization as showed in this section (Section 4.2.2).

4.2.1 Summarizing a Family by its Super Local Transitions

When trying to construct a CA β from a CA α and a local mapping h from the families of space-time diagrams as suggested by the formal definitions, there is huge amount of redundancy. All entries of the local transition relation δ_{Φ_h} appear many times in Φ_h, each of them being produced from the same recurring patterns in the space-time diagrams of α. In fact, it is more efficient to simply collect these recurring patterns that we may call *super local transitions*, and work from them without constructing Φ_h at all. It is specially useful because we consider a huge number of local mappings from a single CA α.

Definition 11. *For a given CA α, the* super local transition table Δ_α *consists of two sets $(\Delta_\alpha)_{\mathtt{z}} \subseteq {\Sigma_\alpha}^3$ and $(\Delta_\alpha)_{\mathtt{s}} \subseteq {\Sigma_\alpha}^5 \times {\Sigma_\alpha}^3$ defined as:*

$$\begin{aligned}(s_{-1}, s_0, s_1) \ \in (\Delta_\alpha)_{\mathtt{z}} &:\Leftrightarrow \exists (d, p) \in \mathrm{D}_\alpha \times \mathbb{Z} \\ &\quad s.t.\ s_i = d(0, p+i), \\ ((s^0_{-2}, s^0_{-1}, s^0_0, s^0_1, s^0_2), (s^1_{-1}, s^1_0, s^1_1)) \in (\Delta_\alpha)_{\mathtt{s}} &:\Leftrightarrow \exists (d, t, p) \in \mathrm{D}_\alpha \times \mathbb{N} \times \mathbb{Z} \\ &\quad s.t.\ s^j_i = d(t+j, p+i)\end{aligned}$$

Once all these patterns collected, it is possible to construct the local transition relation δ_{Φ_h} as specified in the following proposition.

Proposition 2. *Let h be a local mapping from a CA α to a set S. The local transition relation δ_{Φ_h} of the family of space-time diagram Φ_h generated by h and the super-local transition function Δ_α of α obey:*

$$\begin{aligned}((l^0_{-1}, l^0_0, l^0_1), l^1_0) \in \delta_{\Phi_h} \ \Leftrightarrow \ & \exists (s_{-1}, s_0, s_1) \ \in (\Delta_\alpha)_{\mathtt{z}} \\ & \quad s.t.\ l^0_i = h_{\mathtt{z}}(s_i) \text{ and } l^1_0 = h_{\mathtt{s}}(s_{-1}, s_0, s_1) \\ \vee \ & \exists ((s^0_{-2}, s^0_{-1}, s^0_0, s^0_1, s^0_2), (s^1_{-1}, s^1_0, s^1_1)) \in (\Delta_\alpha)_{\mathtt{s}} \\ & \quad s.t.\ l^j_i = h_{\mathtt{s}}(s^j_{i-1}, s^j_i, s^j_{i+1})\end{aligned}$$

Proof

This is obtained by taking the defining property of δ_D given in Def 6, particularize it for $D = \Phi_h$ using Def 10, and rewriting it in terms of the defining properties of the components of the super-local transition given in Def 11.

We now have an efficient way to build the local transition relation δ_{Φ_h}. When it is functional, it determines a cellular automaton $\beta = \Gamma_{\Phi_h}$. For our purpose, we need to test or ensure in some way that β is an FSSP solution.

4.2.2 Local Mappings and FSSP

We first note that the constraints put by the FSSP on space-time diagrams induces constraints on local simulations between FSSP solutions. So we can restrict our attention to local mappings respecting these constraints as formalized by the following definition and proposition.

Definition 12. *A local mapping h from an FSSP solution α to the states Σ_β of an FSSP-candidate CA β is said to be* FSSP-compliant *if it is such that:*

- $h_{\mathtt{z}}$ *maps* $\star_\alpha$, $\mathtt{G}_\alpha$, *and* $\mathtt{Q}_\alpha$ *respectively to* $\star_\beta$, $\mathtt{G}_\beta$, *and* $\mathtt{Q}_\beta$,
- $h_{\mathtt{s}}(l_{-1}, l_0, l_1) = \star_\beta$ *if and only if* $\delta_\alpha(l_{-1}, l_0, l_1) = \star_\alpha$ *(meaning simply* $l_0 = \star_\alpha$*),*
- $h_{\mathtt{s}}(l_{-1}, l_0, l_1) = \mathtt{F}_\beta$ *if and only if* $\delta_\alpha(l_{-1}, l_0, l_1) = \mathtt{F}_\alpha$, *and*
- $h_{\mathtt{s}}(\mathtt{Q}_\alpha, \mathtt{Q}_\alpha, \mathtt{Q}_\alpha) = h_{\mathtt{s}}(\mathtt{Q}_\alpha, \mathtt{Q}_\alpha, \star) = \mathtt{Q}_\beta$.

Proposition 3. *Let α be an FSSP solution CA, β an FSSP-candidate CA and h a local simulation from α to β. If β is an FSSP solution with the same synchronization times as α, then h is FSSP-compliant.*

Proof

Indeed, if both α and β are FSSP-solutions with the same synchronization times, then there are many parts of the space-time diagrams that are essentially the same: the initial configuration, the border part, the firing part and the quiescent part. The fours parts leads to the four properties of the proposition, taking into account Def 10 of local simulation and Prop 1.

The following proposition is at the same time not difficult once noted, but extremely surprising and useful: the simple constraints above are also "totally characterizing" and the previous implication is in fact an equivalence. This means in particular that it is not necessary to generate space-time diagrams to check if a constructed CA is an FSSP solution, which saves lots of computations.

Proposition 4. *Let α be an FSSP solution, β an FSSP-candidate CA and h be local simulation from α to β. If h is FSSP-compliant, then β is an FSSP solution with the same synchronization times as α.*

Proof

To see this, consider a diagram $d \in \mathrm{D}_\alpha$ of the solution α. Since α is an FSSP solution, the four characteristic part of d are again the initial configuration, the border part, the firing part and the quiescent part. Since h is FSSP-compliant, it reproduces these parts in the diagrams of β. But these parts of enough to conclude that β is as stated.

4.3 Exploring The Graph of Local Mappings

4.3.1 The Graph of FSSP-compliant Local Mappings

In our actual algorithm, we take as input an existing FSSP solution α and fix a set of state S of size $\mid \Sigma_\alpha \mid$. The search space consists of all FSSP-compliant local mappings from α to S, the neighbors $\mathrm{N}(h)$ of a local mapping h being all h' that differ from h on exactly one entry, *i.e.* $\mathrm{N}(h) := \{h' \mid \exists!(l_{-1}, l_0, l_1) \in \mathrm{dom}(\delta_\alpha) \text{ s.t. } h_{\mathtt{s}}(l_{-1}, l_0, l_1) \neq h'_{\mathtt{s}}(l_{-1}, l_0, l_1)\}$. More precisely, the mappings are considered modulo bijections of S. Indeed, two mappings h and h' are considered equivalent if there is some bijection $r : S \to S$ such that $h_{\mathtt{z}} = r \circ h'_{\mathtt{z}}$ and $h_{\mathtt{s}} = r \circ h'_{\mathtt{s}}$. So the search space is, in a sense, made of equivalence classes, each class being represented by a particular element. This element is chosen to be the only mapping h in the class such that $h_{\mathtt{s}}$ is monotonic according to arbitrary total orders on $\mathrm{dom}(\delta_\alpha)$ and S fixed for the entire run of the algorithm.

Considering 6-states solutions, let us denote $\Sigma_\alpha = \{\star_\alpha, \mathtt{G}_\alpha, \mathtt{Q}_\alpha, \mathtt{F}_\alpha, \mathtt{A}_\alpha, \mathtt{B}_\alpha, \mathtt{C}_\alpha\}$ and $S = \{\star, \mathtt{G}, \mathtt{Q}, \mathtt{F}, \mathtt{A}, \mathtt{B}, \mathtt{C}\}$[1]. In each of these sets, four of the states are the special FSSP solution states (Def. 9 and Def. 4). Only the three states `A`, `B`, `C` come with no constraints. We can thus evaluate the size of the search space by looking at the degrees of freedom of FSSP-compliant local mappings (Def. 12).

Indeed, looking at Def 12 shows that all FSSP-compliant local mappings h from α to S have the same partial function $h_{\mathtt{z}}$, and the same value $h_{\mathtt{s}}(l_{-1}, l_0, l_1)$ for those entries $(l_{-1}, l_0, l_1) \in \mathrm{dom}(\delta_\alpha)$ forced to $\star$, `Q` or `F`. For all other entries (l_{-1}, l_0, l_1), $h_{\mathtt{s}}(l_{-1}, l_0, l_1)$ cannot take the values $\star$ nor `F`, leaving 5 values available. So given an initial solution α, the number of local mappings is 5^x where x is the size of $\mathrm{dom}(\delta_\alpha)$ without those entries constrained in Def. 12. To give an idea, for the solution 668 of the 718 solutions, $x = 86$ to the size of the search has 61 digits, and for Mazoyer's solution, $x = 112$ leading to a number with 79 digits.

4.3.2 Preparation Before the Algorithm

As described in Section 4.2.1, the local mappings are evaluated from the super local transition table. To build this table, we generate, for each size n from 2 to 5000, the space-time diagram $\mathrm{D}_\alpha(\bar{n})$ and collect all super local transitions occurring from time 0 to $2n-4$ and from position 1 to n. Note that for all known minimum-time 6-state solutions, no additional super local transitions appear after $n = 250$.

The starting point of the exploration is the local mapping h_α corresponding to the local transition function δ_α itself, *i.e.* $(h_\alpha)_{\mathtt{z}} = \rho \restriction \{\star_\alpha, \mathtt{G}_\alpha, \mathtt{Q}_\alpha\}$ and $(h_\alpha)_{\mathtt{s}} = \rho \circ \delta_\alpha$ for the obvious bijection $\rho : \Sigma_\alpha \to S$. This local mapping is obviously FSSP-compliant since it is local simulation from α to α.

4.3.3 The Exploration Algorithm

To explain the algorithm, let us first consider the last parameter k to be 0, so that line 7 of the `explore` algorithm can be considered to be simply $S \leftarrow \mathrm{N}(h)$. In this case, the algorithm starts with h_α, and explores its neighbors to collect all local simulations. Then the neighbors of those local simulations are considered to collect more local simulations, and so on so forth. In other words, the connected component of the sub-graph consisting only of the local simulations is collected. More precisely, the variable H collects all local simulations, $H_{current}$ contains the simulation discovered in the previous round and whose neighbors should be examined in current round, and the newly discovered local simulations are put in H_{next} for the next round. The function `isSimul` uses the super local transition table to construct the local transition relation of Φ_h and check if it is functional, *i.e.* if it is a local transition function of a valid CA Γ_{Φ_h}. By our construction, a valid CA is necessarily an FSSP solution making this operation really cheap.

When $k > 0$, line 7 puts in S not only $N(h)$ but also $N(h')$ for some local *mapping* h' obtained by k random modifications of h. The hope is to jump to another connected component of the "local simulation sub-graph". Since h' could be a (new or already considered) local *simulation*, a quick check is necessary.

[1] Recall that we do not count the $\star$ states.

Algorithm 1:

```
1  explore(Δ_α, h_α, k)
2     H ← {h_α}
3     H_current ← {h_α}
4     while |H_current| > 0 do
5        H_next ← {}
6        for h ∈ H_current do
7           S, H ← pertN(Δ_α, H, h, k)
8           for h' ∈ (S \ H) do
9              if isSimul(h', Δ_α) then
10                H_next ← H_next ∪ {h'}
11                H ← H ∪ {h'}
12             end
13          end
14       end
15       H_current ← H_next
16    end
17    return H
```

Algorithm 2:

```
1  pertN(Δ_α, H, h, k)
2     S ← N(h)
3     h' ← perturbation(h, k)
4     if h' ∉ H then
5        S ← S ∪ N(h')
6        if isSimul(h', Δ_α) then
7           H ← H ∪ {h'}
8        end
9     end
10    return S, H
```

4.4 Analyzing the Results

4.4.1 Analyzing the Local Simulations

To find more FSSP solutions, we implemented many algorithms, gradually simplifying them into the one presented in this paper. It has been run on an Ubuntu Marvin machine with 32 cores of 2.00GHz speed and 126Gb of memory. However, the implementation being sequential, only two cores was used by the program. The original plan was to generate as many solutions as possible but we had some problems with the management of quotas in the shared machine. So we only expose some selected data to show the relevance of the approach.

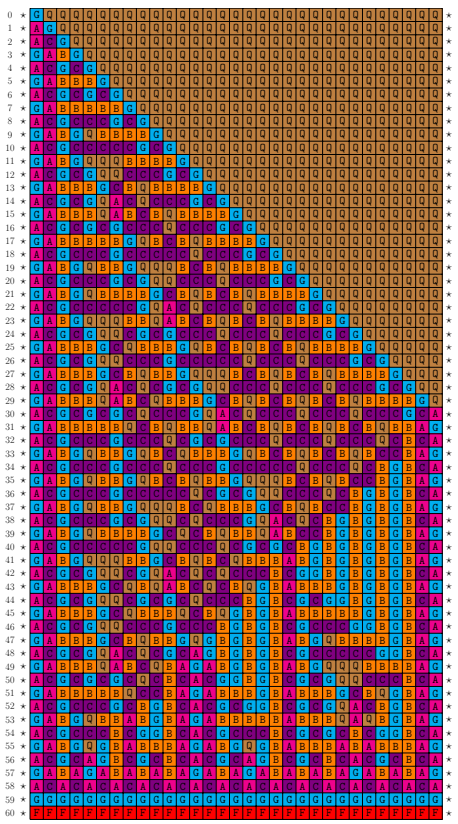
When running the program with the solution 355 and $k = 0$, the program used 14Gb of memory and stopped after 27.5 hours and found 9,584,134 local simulations! A second run of the program for this solution with $k = 3$ found 11,506,263 local simulations after 80.5 hours. This indicates that perturbations are useful but the second run find only 1,922,129 additional local simulations but its computing time is three times more than the first run. Testing whether a local simulation belongs to set H obviously takes more and more time as more local mappings are discovered but there might be some understanding to gain about the proper mapping landscape too in order to improve the situation.

The transition table for the original Mazoyer's solution can be found in [21], but also in [33] together with other minimal-time solution transition tables. When running the program of the original Mazoyer's solution with different values of k we obtained the following number of new solutions for different runs.

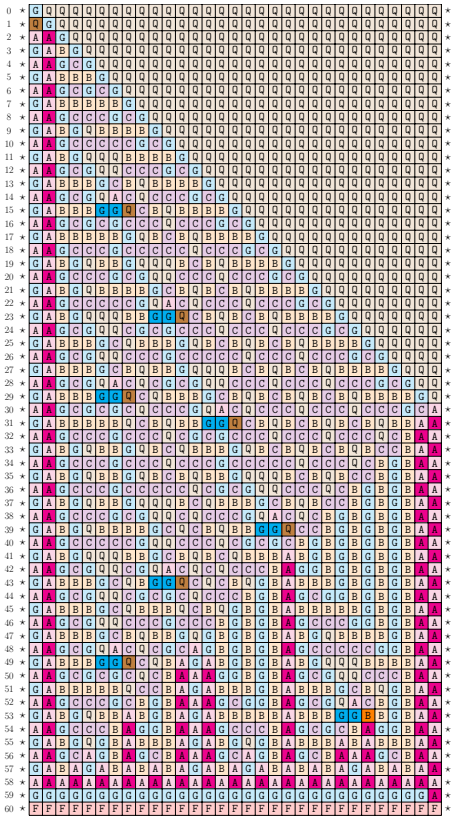
k	number of solutions found by 10 different runs
0	**644**
1	20682, 17645, 20731, 16139, **20731**, 9538, 20626, 20682, 20054, 20490
2	9451, 9451, 20595, 8241, **37275**, 3817, 17421, 8241, 17317, 19895
3	644, 644, 644, 644, 644, 644, 644, 731, 8241, **8241**
4	644, 644, 644, 2908, 644, 644, 644, 644, 644, **8241**
5	644, 644, 644, 644, 644, 644, 644, 644, 644, 644

The behavior with $k = 1$ seems to be pretty robust, but the bigger number of results is obtained with $k = 2$. For $k \geq 3$, the perturbations seem to be too violent and do not generally lead to more solutions. Note that while the solutions do not have less states, the number of transitions do change. We show in Figure 4.1a the solution 668 (the only Mazoyer-like solutions found among the 718 solutions), and one of its simulations having less transitions in Figure 4.1b. For fun, we also show in Figure 4.1d a local simulation of the solution 355 having alternating states at time $2n-3$, illustrating how local simulation rearrange locally the information. The identical part is represented with lighter colors to highlight the differences.

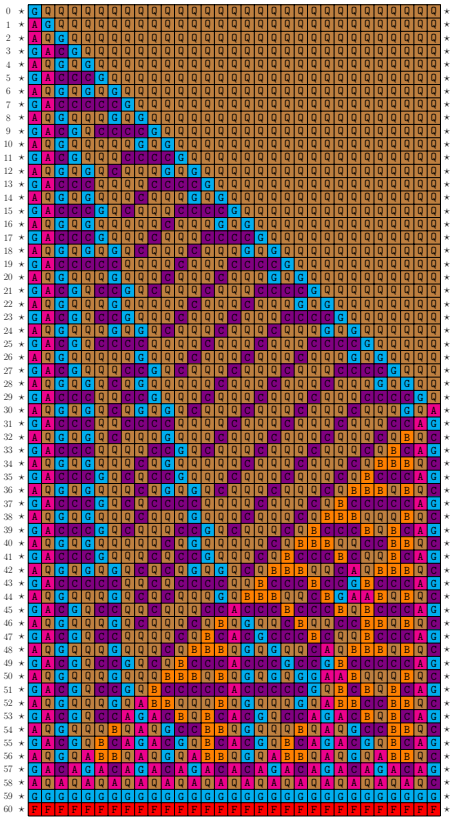
Proposition 5. *There are at least 11,506,263 minimum-time 6-state FSSP solutions.*



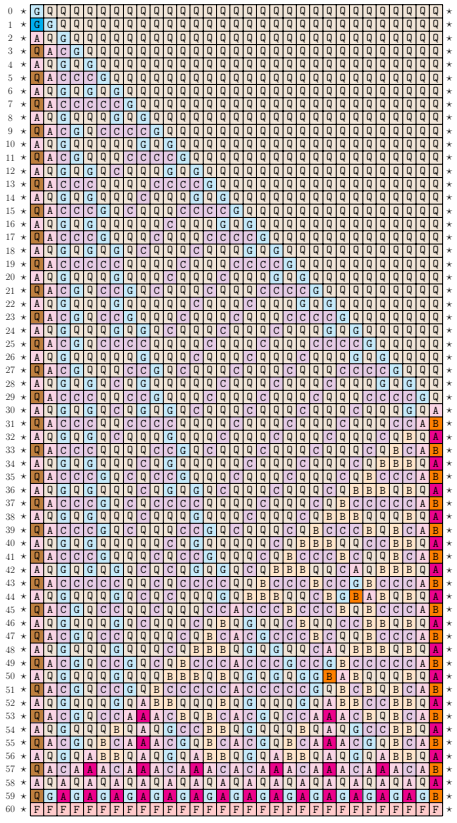
(a) Original solution 668: 93 rules



(b) A local simulation of 668: 90 rules



(c) Original solution 355: 101 rules



(d) A local simulation of 355: 102 rules

Figure 4.1: Some FSSP space-time diagrams of size 31

Chapter 5

Beyond Synchronization Problems

Up to now, we have only considered FSSP solutions, but the approach is more general. It is possible to formalize different classes of problems to which the results generalizes. Informally, they correspond to problems specified in terms of space-time constraints. In the following, we proceed more explicitly. We take a specific actively studied example, the n^3-RTSG already described in Chapter 1 and show why and how things adapt naturally. In fact, the content of this chapter applies readily to any S-RTSG problem, for any sequence S.

In Section 5.1, we define formally RTSG candidate and solutions and recall the solution of Kamikawa and Umeo (6 states and 74 transition rules, state of the art for the n^3 sequence) with enough detail for the results to be reproducible. In the following sections, we describe RTSG-compliant local mappings and use the tools developed in the previous chapter to obtain a new solution using only 4 states and 55 transition rules.

5.1 Real-Time Sequence Generators, Formally

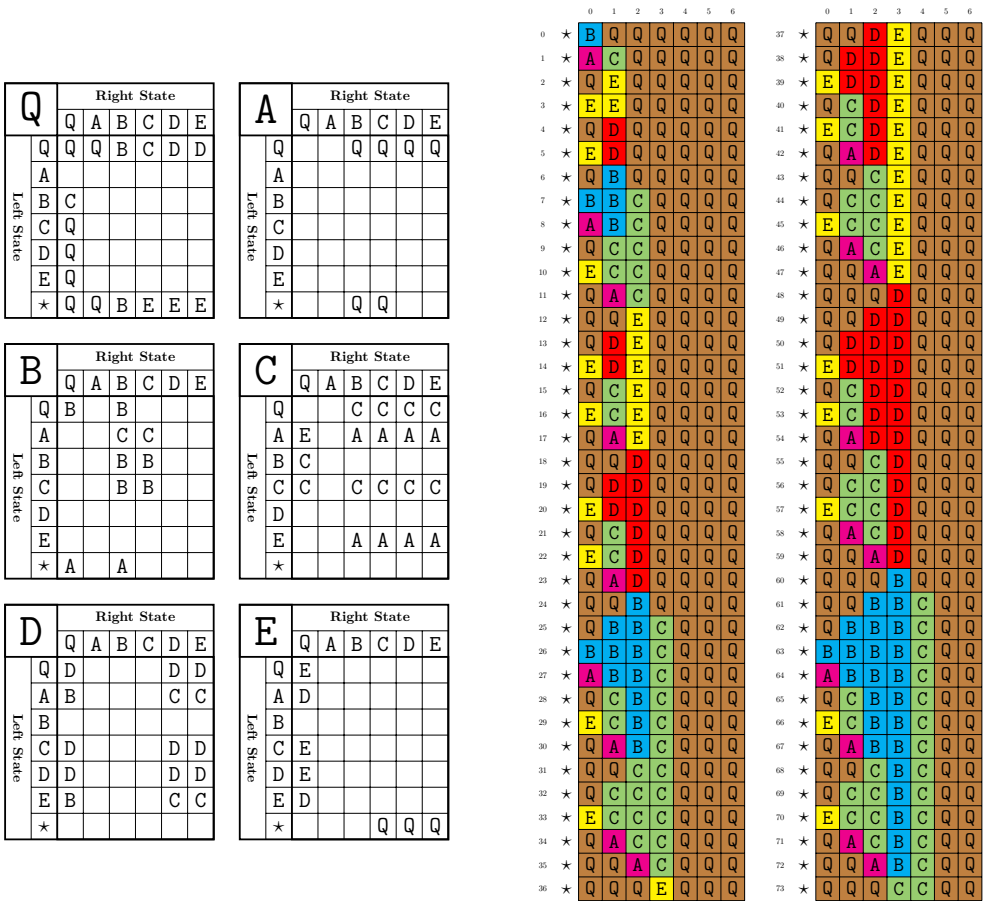
The following definitions are obtained by adapting the FSSP definitions 9 and 4. Where FSSP has an infinite number of finite initial configurations, RTSG as a single infinite initial configuration.

Definition 13. *A cellular automaton is* RTSG-candidate *if there are four special states* $\star_\alpha, \mathtt{B}_\alpha, \mathtt{Q}_\alpha, \mathtt{S}_\alpha \in \Sigma_\alpha$ *satisfying the following conditions.*

1. $\star_\alpha$ *is an* outside state*,* i.e. $\forall\, l \in dom(\delta_\alpha)$*,* $\delta_\alpha(l) = \star_\alpha \Leftrightarrow l(0) = \star_\alpha$*.*
2. $\mathtt{B}_\alpha$ *is the* launching state*,* i.e. $\mathrm{I}_\alpha = \{\overrightarrow{\infty}_\alpha\}$ *with* $\overrightarrow{\infty}_\alpha$ *being the* RTSG initial configuration *defined as* $\overrightarrow{\infty}_\alpha(p) = \star_\alpha$*,* $\mathtt{B}_\alpha$ *and* $\mathtt{Q}_\alpha$ *for* $p < 0$*,* $p = 0$ *and* $p \geq 1$ *respectively.*
3. $\mathtt{Q}_\alpha$ *is a* quiescent state*,* i.e. $\delta_\alpha(\mathtt{Q}_\alpha, \mathtt{Q}_\alpha, \mathtt{Q}_\alpha) = \mathtt{Q}_\alpha$ *on one hand, and* $\delta_\alpha(\star_\alpha, \mathtt{Q}_\alpha, \mathtt{Q}_\alpha) = \mathtt{Q}_\alpha$ *if it is defined, on the the hand.*

As for FSSP solutions and FSSP candidates, RTSG candidates are almost S-RSTG solutions, except that they lack the last condition of Def 5.

Proposition 6. *There is a* n^3*-RTSG solution using 6 states and 74 transitions.*



Q	Right State					
Left State	Q	A	B	C	D	E
Q	Q	Q	B	C	D	D
A						
B	C					
C	Q					
D	Q					
E	Q					
⋆	Q	Q	B	E	E	E

A	Right State					
Left State	Q	A	B	C	D	E
Q			Q	Q	Q	Q
A						
B						
C						
D						
E						
⋆			Q	Q		

B	Right State					
Left State	Q	A	B	C	D	E
Q	B		B			
A			C	C		
B			B	B		
C			B	B		
D						
E						
⋆	A		A			

C	Right State					
Left State	Q	A	B	C	D	E
Q			C	C	C	C
A	E		A	A	A	A
B	C					
C	C		C	C	C	C
D						
E			A	A	A	A
⋆						

D	Right State					
Left State	Q	A	B	C	D	E
Q	D				D	D
A	B				C	C
B						
C	D				D	D
D	D				D	D
E	B				C	C
⋆						

E	Right State					
Left State	Q	A	B	C	D	E
Q	E					
A	D					
B						
C	E					
D	E					
E	D					
⋆				Q	Q	Q

Figure 5.1: Transition table and space-time diagram of Kamikawa and Umeo's solution $\mathfrak{KU}^6_{74}$ using 6 state and 74 transitions.

Proof

In Figure 5.1 is the solution of Kamikawa and Umeo, reproduced with the same format as their paper to ease comparison. The set of state is $\Sigma_{\mathfrak{KU}^6_{74}} = \{\star, Q, A, B, C, D, E\}$, the special states being $\star_{\mathfrak{KU}^6_{74}} = \star$, $\mathtt{B}_{\mathfrak{KU}^6_{74}} = B$, $\mathtt{Q}_{\mathfrak{KU}^6_{74}} = Q$, and $\mathtt{S}_{\mathfrak{KU}^6_{74}} = A$. Also, the local transition function $\delta_{\mathfrak{KU}^6_{74}}$ contains the above entries and additional obvious entries for the outside state $\star$. The proof of correction can be found in [12].

The space-time diagram of this solution is depicted up in the two right columns of Figure 5.1, where the cell at position 0 has the state A at time 1, 8, 27, and 64 as expected. In the original paper [12], note that table of D wrongly has column C filled with the content of column E. This mistake is easy to catch by examining the proofs and space-time diagrams of the paper.

Now let us consider local mappings and local simulations for RTSG. The following should be compared to their FSSP counterpart (Definition 12).

Definition 14. *A local mapping ℓ from an RTSG-candidate α to the states Σ_β of an RTSG-candidate CA β is said to be* RTSG-compliant *if it is such that*

- $\ell_{\mathtt{z}}$ *maps* $\star_\alpha$, $\mathtt{B}_\alpha$, *and* $\mathtt{Q}_\alpha$ *respectively to* $\star_\beta$, $\mathtt{B}_\beta$, *and* $\mathtt{Q}_\beta$,

- $\ell_{\mathtt{s}}(c_{-1}, c_0, c_1) = \star_\beta$ *if and only if* $\delta_\alpha(c_{-1}, c_0, c_1) = \star_\alpha$ *(meaning simply* $c_0 = \star_\alpha$*),*
- $\ell_{\mathtt{s}}(\star_\alpha, c_0, c_1) = \mathtt{S}_\beta$ *if and only if* $\delta_\alpha(\star_\alpha, c_0, c_1) = \mathtt{S}_\alpha$*, and*
- $\ell_{\mathtt{s}}(\mathtt{Q}_\alpha, \mathtt{Q}_\alpha, \mathtt{Q}_\alpha) = \mathtt{Q}_\beta$ *and* $\ell_{\mathtt{s}}(\star_\alpha, \mathtt{Q}_\alpha, \mathtt{Q}_\alpha) = \mathtt{Q}_\beta$ *if it is defined.*

Proposition 7. *Given a sequence* $S \subseteq \mathbb{N}$*, let* α *be an S-RTSG solution CA,* β *an S-RTSG-candidate CA and* ℓ *a local simulation from* α *to* β*.* β *is an S-RTSG solution if and only if* ℓ *is RTSG-compliant.*

Proof

To see this, consider the diagram $d \in \mathrm{D}_\alpha$ of the solution α. The special RTSG states appear at specific places and ℓ ensures or witnesses, depending on the direction of the implication considered, that these special states/places are conserved in $\ell(d) \in \mathrm{D}_\beta$, (Definition 10). Indeed, condition (1) is just about the initial configuration, condition (2) is about the conservation of the outside state, condition (3) is about the conservation of the special generation state for the left-most cell only and condition (4) about the conservation of the quiescent state behaviour. These conditions are sufficient to ensure that β is a solution, and clearly necessary since they perfectly match Definitions 13 and 5 of the problem.

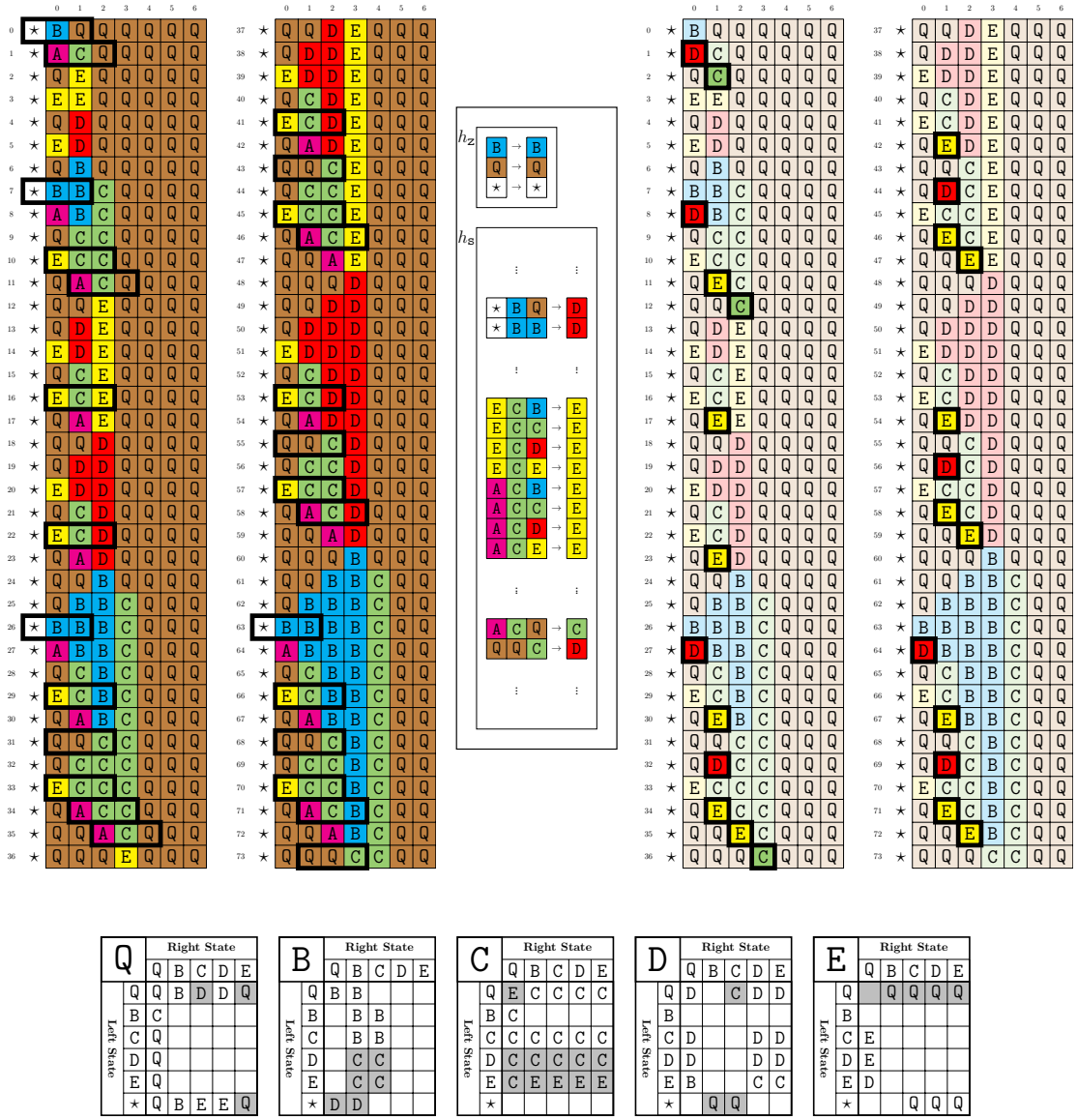
Let us repeat this important remark. Once α is fixed, β can be reconstructed from ℓ, and ℓ from β. So the local mapping ℓ is just another representation of the RTSG-candidate β that it generates (see Definition 10), but it is much easier to check the compliance of the local mapping than the correction of CA β as an RTSG solution, and this is the key property than justifies this particular application of local mappings.

5.2 A Hand-Crafted Local Simulation from $\mathfrak{KU}^6_{74}$ (to $\mathfrak{MM}^5_{72}$)

Now that we have the relevant definitions, the study begins by first noting that the space-time diagram of solution $\mathfrak{KU}^6_{74}$, as depicted in Fig. 5.1, make a very sparse use of state A. It is subtle task to remove this state directly at the level of the transition table. The task is however very direct using local mappings.

Similarly to the exploration algorithm, the first local mapping that we consider is the identity local mapping *id* given by the local transition function of the 6-state solution of Proposition 6. This local mapping simply transforms this solution into itself. The point here is that we can now work with the local mapping because each modification in the local mapping corresponds directly to a uniform set of modifications in the space-time diagram which may or may not be deterministic after these modifications.

Let us now describe how the second, hand-crafted, local mapping is obtained. The goal is to remove the state A. This means changing every entry (x, y, z) of $id_{\mathtt{s}}$



Q	Right State				
Left State	Q	B	C	D	E
Q	Q	B	D	D	Q
B	C				
C	Q				
D	Q				
E	Q				
⋆	Q	B	E	E	Q

B	Right State				
Left State	Q	B	C	D	E
Q	B	B			
B		B	B		
C		B	B		
D		C	C		
E		C	C		
⋆	D	D			

C	Right State				
Left State	Q	B	C	D	E
Q	E	C	C	C	C
B	C				
C	C	C	C	C	C
D	C	C	C	C	C
E	C	E	E	E	E
⋆					

D	Right State				
Left State	Q	B	C	D	E
Q	D		C	D	D
B					
C	D			D	D
D	D			D	D
E	B			C	C
⋆		Q	Q		

E	Right State				
Left State	Q	B	C	D	E
Q		Q	Q	Q	Q
B					
C	E				
D	E				
E	D				
⋆			Q	Q	Q

Figure 5.2: 6-state diagram, hand-crafted local mapping, the resulting 5-state diagram, and its transition table using 72 transitions

such that $id_{\mathsf{s}}(x, y, z) = A$. But since A is the special generating state, we can not replace it by B, Q or E since they already appear in the evolution of the leftmost cell. So we can consider either C or D. However, looking at time 1, we see that changing A into C would lead to a CCQ local configuration, which is already used. So we heuristically choose D instead, to have DCQ at time 1, an unused local configuration. To summarize, for the leftmost cell we choose to change A by D, and for the other cells, we can choose any state *a priori*.

The second local mapping is thus obtained by taking every local configurations (x, y, z) of id_{s} such that $id_{\mathsf{s}}(x, y, z) = A$, and setting them to D if $x = \star$, and to E otherwise. The result is not a deterministic space-time diagram, but this is easily corrected with two additional modifications for ACQ and QQC, leading to the local mapping depicted in the center of Figure 5.2. The space-time diagram on the right is obtained by applying the local mapping on the space-time on the left as indicated by the outlined local configuration on the left, and resulting state on the right, at

the following timestep in direct application of Definition 10.

Proposition 8. *There is a n^3-RTSG solution using 5 states and 72 transitions.*

Proof

First note that the right space-time diagram of Fig. 5.2 is deterministic. We can therefore extract the solution $\mathfrak{MN}^5_{72}$ whose transition table is given in Figure 5.2. Its set of state is $\Sigma_{\mathfrak{MN}^5_{72}} = \{\star, Q, B, C, D, E\}$ and its special generation state is $\mathtt{S}_{\mathfrak{MN}^5_{72}} = D$. No additional transitions appear after the space-time shown in Figure 5.2. To prove this CA to be an n^3-RTSG solution, it is enough to check that the local mapping is RTSG-compliant by examining the 12 entries in Fig. 5.2, and since the source CA is an n^3-RTSG solution, we can conclude using Proposition 7.

To ease the comparison of this 5-state solution with the original 6-state solution, the transitions that are different, added or removed are highlighted in the above table. Of course, all transition containing A should be considered as removed. The reader can check that these differences do not correspond exactly to those described in the local mapping.

5.3 Automatic Explorations from $\mathfrak{MN}^5_{72}$

5.3.1 Automatic Exploration from $\mathfrak{MN}^5_{72}$ (to $\mathfrak{MN}^5_{58}$)

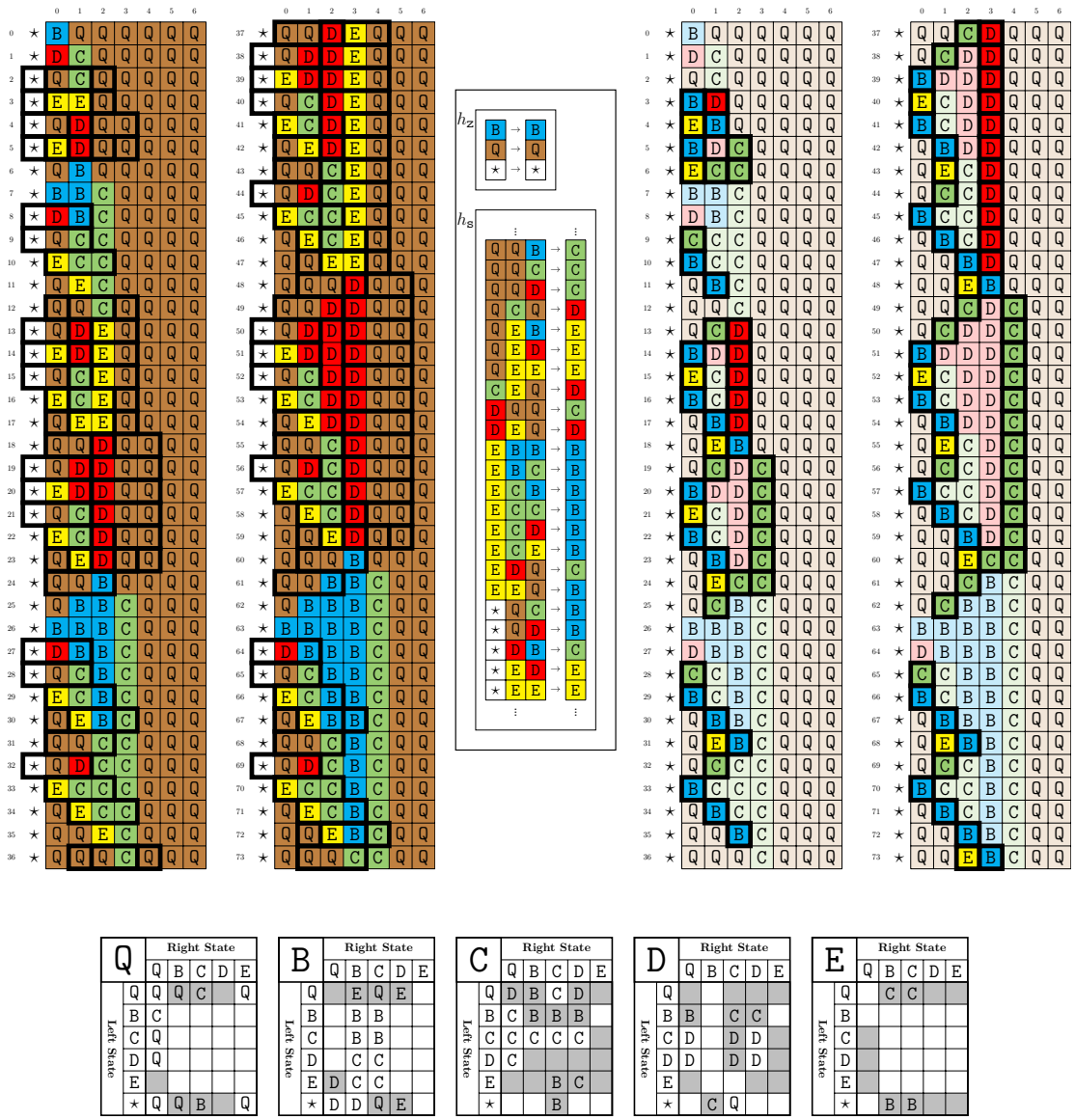
Now that we have a first 5-state solution, we are ready to generate millions of them.

Running the algorithm on a 32 cores of 2.00GHz machine having 126Gb of memory, we obtain so many solutions that the algorithm stops because it runs out of memory resource. The first time, we ran the algorithm with $k = 0$. The program actually uses 2 cores and about 43 Gb of memory. We did not optimize the program nor did we check the configuration of the Java Virtual Machine for this Java implementation. Since the machine is shared, the following data are not really reproducible, but gives an idea of the execution.

- after 1 days, about 15 millions local simulations.
- after 6 days, about 85 millions local simulations.
- after 20 days, about 90 millions local simulations.

The number of solutions found each day was steady for the 6 firsts days then dropped, presumably because of memory issues. Running concurrently the program with $k = 2$, it uses 2 cores and 36 Gb of memory before it stops because of the same lake of memory.

- after 1 days, about 15 millions local simulations.
- after 6 days, about 70 millions local simulations.



Q	Right State				
Left State	Q	B	C	D	E
Q	Q	Q	C		Q
B	C				
C	Q				
D	Q				
E					
⋆	Q	Q	B		Q

B	Right State				
Left State	Q	B	C	D	E
Q		E	Q	E	
B		B	B		
C		B	B		
D		C	C		
E	D	C	C		
⋆	D	D	Q	E	

C	Right State				
Left State	Q	B	C	D	E
Q	D	B	C	D	
B	C	B	B	B	
C	C	C	C	C	
D	C				
E			B	C	
⋆			B		

D	Right State				
Left State	Q	B	C	D	E
Q					
B	B		C	C	
C	D		D	D	
D	D		D	D	
E					
⋆		C	Q		

E	Right State				
Left State	Q	B	C	D	E
Q		C	C		
B					
C					
D					
E					
⋆		B	B		

Figure 5.3: Hand-crafted 5-state diagram, optimized 5-state diagram using 58 transitions, and transition table of the latter

- after 20 days, about 74 millions local simulations.

In fact, we had to keep in memory all the solutions and check whether we obtain new solutions up to permutations, in order to be able to have a total number of generated solutions. Better strategies can be found if the goal is only to optimize the solution.

Proposition 9. *There are at least 90,000,000 n^3-RTSG solutions using 5 states.*

Among these millions of solutions, no 4-state solutions are found, but 32379 of them have fewer transitions. In the following table, the first line indicates a number of transition and the second line the number of solutions having this number of transitions.

Proposition 10. *There is a n^3-RTSG solution using 5 states and 58 transitions.*

58	59	60	61	62	63	64	65	66	67	68	69	70	71
1	7	22	51	98	174	336	589	1044	1618	2696	4643	7671	13429

Proof

The transition table of the generated solution $\mathfrak{MN}^5_{58}$ is shown in Figure 5.3. The 23 entries differing from the identity local mapping are also depicted. It is then a matter of checking that they are compliant and apply Proposition 7 to conclude as before.

5.3.2 Automatic Exploration from $\mathfrak{MN}^5_{58}$ (to $\mathfrak{MN}^4_{55}$)

Now, from $\mathfrak{MN}^5_{58}$, we collected his super local transition table up to time $t = 100^3$. There are 264 super local transitions and the last super local transition appears at time $t = 235$. We explore $\mathfrak{MN}^5_{58}$ using our algorithm with $k = 0$ in the same machine Ubuntu, we found 67925109 solutions after 280935 second within 13788797329 evaluations. This time is not running out of memory resource. Among the solutions found, there are 3 5-state 55-rule solutions, 3 5-state 56-rule solutions, 13 5-state 57-rule solutions and only 1 4-state 55-rule solution as we denote $\mathfrak{MN}^4_{55}$. Fig 5.4 shows a diagram of $\mathfrak{MN}^4_{55}$ in the two columns in the right up to time $t = 73$ and its local transition table.

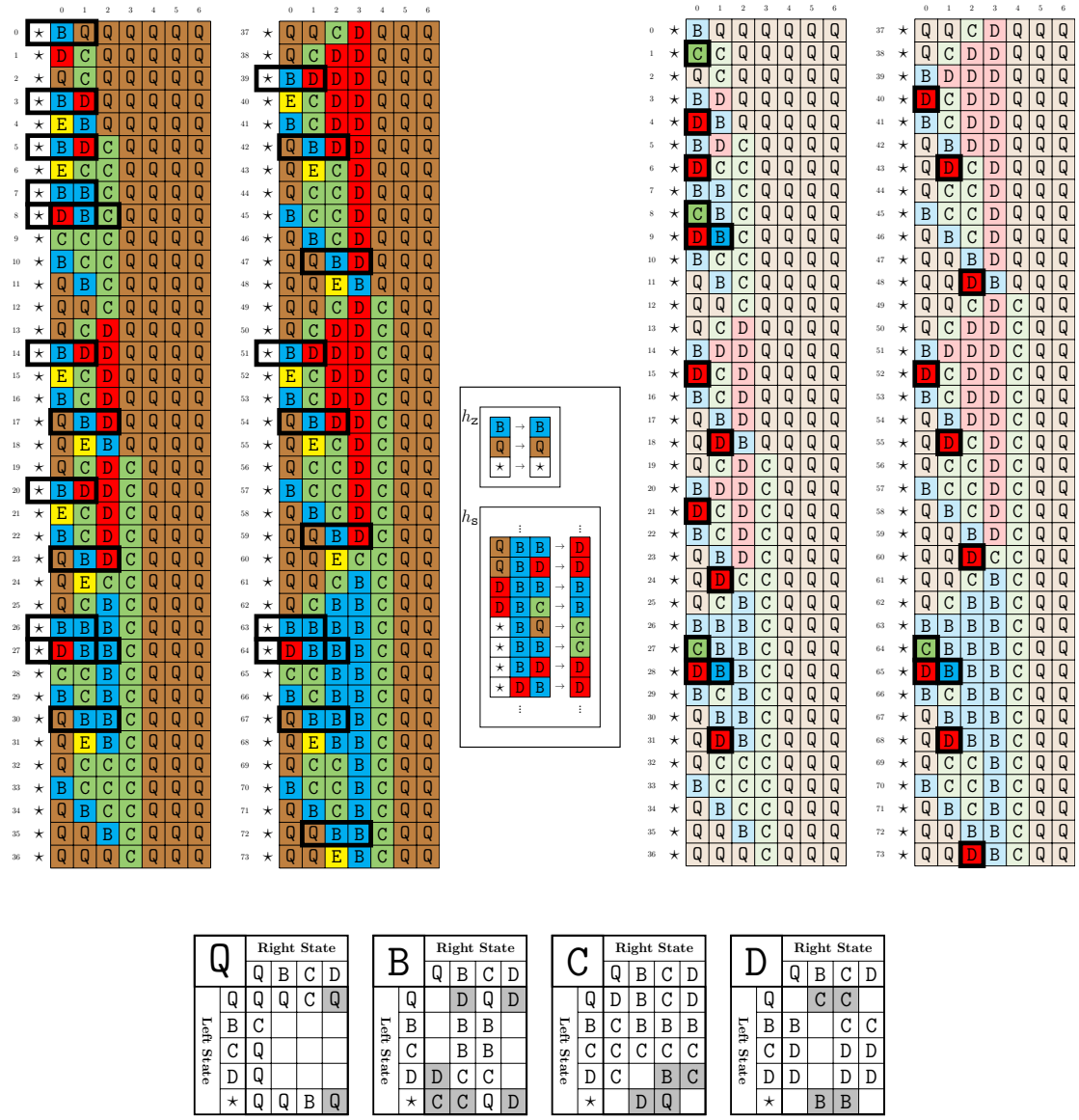
Proposition 11. *There is a n^3-RTSG solution using 4 states and 55 transitions.*

We tried to explore $\mathfrak{MN}^4_{55}$ but no local simulation found. We found that $\mathfrak{MN}^4_{55}$ is a reduction of $\mathfrak{KU}^6_{74}$ and $\mathfrak{MN}^5_{72}$ with $dt = 1$ but there is not local simulation from $\mathfrak{MN}^4_{55}$ to $\mathfrak{KU}^6_{74}$ or $\mathfrak{MN}^5_{72}$ with the same $dt = 1$.

5.4 Conclusion

It should be clear by now that the approach can be applied to a large class of problems. For example, the same algorithm used here for the n^3-RTSG problem is not particular to the n^3 sequence and can be used for any sequence S, as clearly indicated in the definitions and propositions above. Also, the slightly differently parameterized algorithm for the minimal-time FSSP is not particular to minimal-time solutions and can be used for any synchronization time, without even specifying this synchronization time to the algorithm. The difference in the parameter only reflects the slightly different notion of compliance for RTSG problems and FSSP. Because the notion of compliance is the only changing factor, the approach can readily be adapted to any class of problem for which an appropriate notion of compliance can be designed. As examplified here, and in the FSSP case, the compliance property is a direct translation of the problem.

There are still many components of this work to communicate properly, including how local mappings compose and relate to each other and how the integration of non-deterministic family of space-time diagrams can allow to explore even more (deterministic) solutions. Beginning these discussions in this conclusion is not necessarily useful. There is nonetheless one aspect on which we should comment. The



Q	Right State			
Left State	Q	B	C	D
Q	Q	Q	C	Q
B	C			
C	Q			
D	Q			
⋆	Q	Q	B	Q

B	Right State			
Left State	Q	B	C	D
Q		D	Q	D
B		B	B	
C		B	B	
D	D	C	C	
⋆	C	C	Q	D

C	Right State			
Left State	Q	B	C	D
Q	D	B	C	D
B	C	B	B	B
C	C	C	C	C
D	C		B	C
⋆		D	Q	

D	Right State			
Left State	Q	B	C	D
Q		C	C	
B	B		C	C
C	D		D	D
D	D		D	D
⋆		B	B	

Figure 5.4: A 5-state diagram using 58 transitions, its reduction 4-state diagram using 55 transitions, and transition table of the latter

notion of local mapping appears to be a bridge between a common practice and a topological tool. Indeed, on the practical side, it is common to work directly at the level of space-time diagrams, and this is this practice that is captured formally, and only partly, by local mappings. This allows to automate this practice. On the other hand, a question was raised about the relation with conjugacy classes, a standard notion in the cellular automata and symbolic dynamics literature [9]. In fact, the concept of local mapping appears to be an adaptation of the notion of shift-equivariant homomorphism between two cellular automaton. Such homomorphisms are usually described on total transition functions, with any configuration being a valid initial configuration. This is a dynamical system point of view not necessarily aligned with the more algorithmic point of view of FSSP and RTSG problems. Local mappings augment the notion of homomorphism by including the partiality of the transition functions and the temporal aspect of the space-time diagrams, essential for the very specification of many algorithmic problem. Forming a bridge

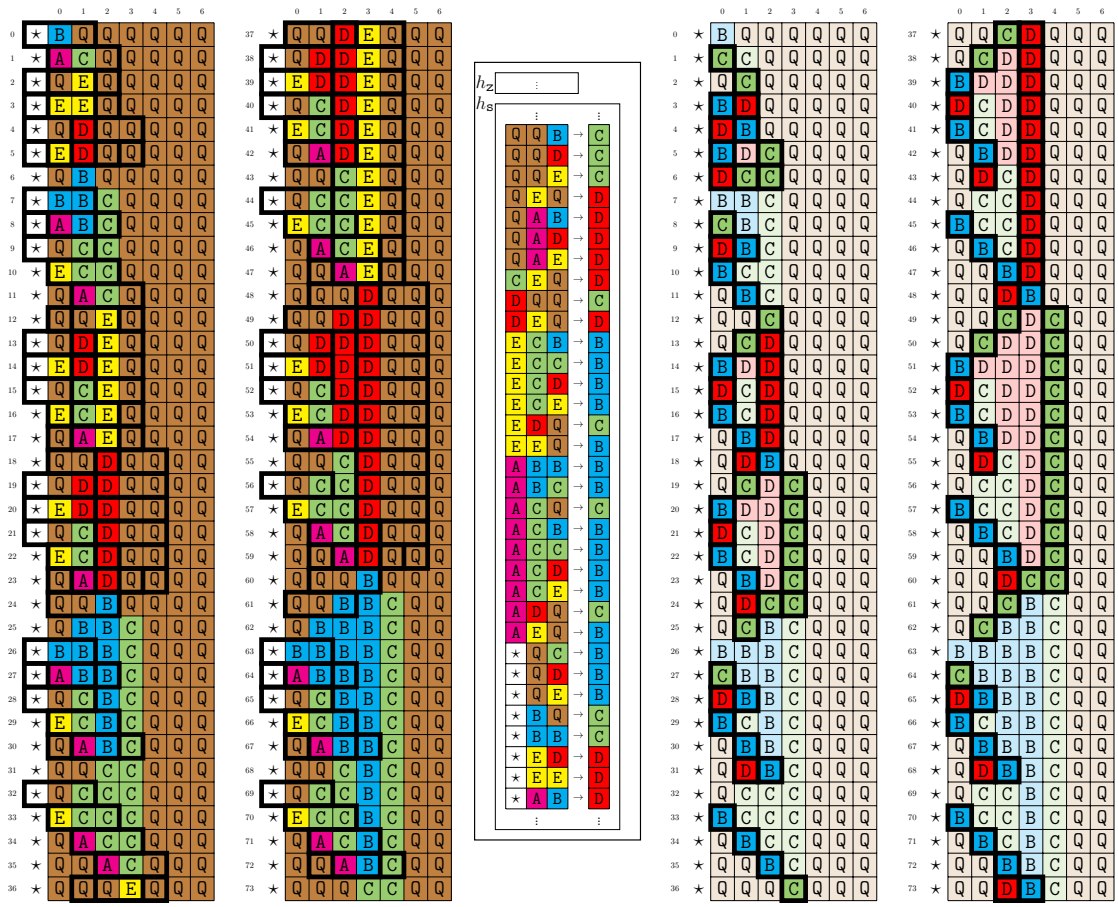


Figure 5.5: Umeo vs 4-state

between the algorithmic and dynamical points of view might be the reason of their effectiveness.

Chapter 6

Final Discussion

There are still lots of study required to fully explore what local mappings and local simulations have to offer. This can be seen very concretely based on questions that naturally arise from the examples considered so far.

A first important concept to further explore is related to the "depth" of the local mappings. Indeed, starting with a CA α, a local simulation produces another CA β. If we start now from β, a local simulation produces yet another CA γ. But γ is, in general, not a local simulation of α. Thinking in terms of space-time diagram, the reason is that each local simulation consult $t-1$ in the origin CA to build time t of the target CA. So γ needs to consult $t-2$ of α. More generally, in order to obtain a kind of transitive closure of the notion of local simulation, we need to consider local simulation of "depth" 2, 3, 4, etc. The composition of a local simulation of depth d_1 with another one of depth d_2 generally leads to a local simulation of depth $d_1 + d_2$. A local simulation of depth 0 is just a quotient of course. All of this creates a nice algebraic structure that should be studied more thoroughly mathematically, but also explored computationally.

The non-mininal-time FSSP solutions $\mathfrak{F}'^{18}_{373}$ to $\mathfrak{U}^6_{78}$ considered in Chapter 2, offers a concrete example. Indeed, although $\mathfrak{F}'^{18}_{373}$ have been created with the same structure as $\mathfrak{U}^6_{78}$, none of these two CA is a local simulation of the other, not at depth 1 at least. In fact, we need to go to depth 2 to see the relation. A depth-2 local mapping h is composed of two initializing function h_0 and h_1 used to create or relate the timesteps 0 and 1 of the target CA from the initial configuration of the source CA, and an additional h_2 used to create or relate the timestep t of the target CA from the timestep $t-2$ of the source CA as illustrated in Figure 6.1. This figure shows the local mapping of depth 2 from $\mathfrak{F}'^{18}_{373}$ to $\mathfrak{U}^6_{78}$. h_0 maps a state of $\mathfrak{F}'^{18}_{373}$ to a state of $\mathfrak{U}^6_{78}$ in the initial configuration. h_1 maps a triplet of $\mathfrak{F}'^{18}_{373}$ in the initial configuration to a state of $\mathfrak{U}^6_{78}$ at time $t = 1$. h_2 maps a quintuplet of $\mathfrak{F}'^{18}_{373}$ at (t, p) to a state of $\mathfrak{U}^6_{78}$ at $(t+2, p)$. We use the colors to illustrate the levels of local mapping.

Another application of this extension is the following. Our guess is that, with a properly large notion of such simulations, it should be possible to classify the 718 solutions into only a few equivalence classes, more or less in two groups: the "mid-way division" solutions and the "two-third division" solutions. Using deeper local simulations indeed reduces the number of equivalence classes down to 19 in our latest unfinished experiments up to depth 6.

But it is clear that there is a need to go beyond this extension local simulations.

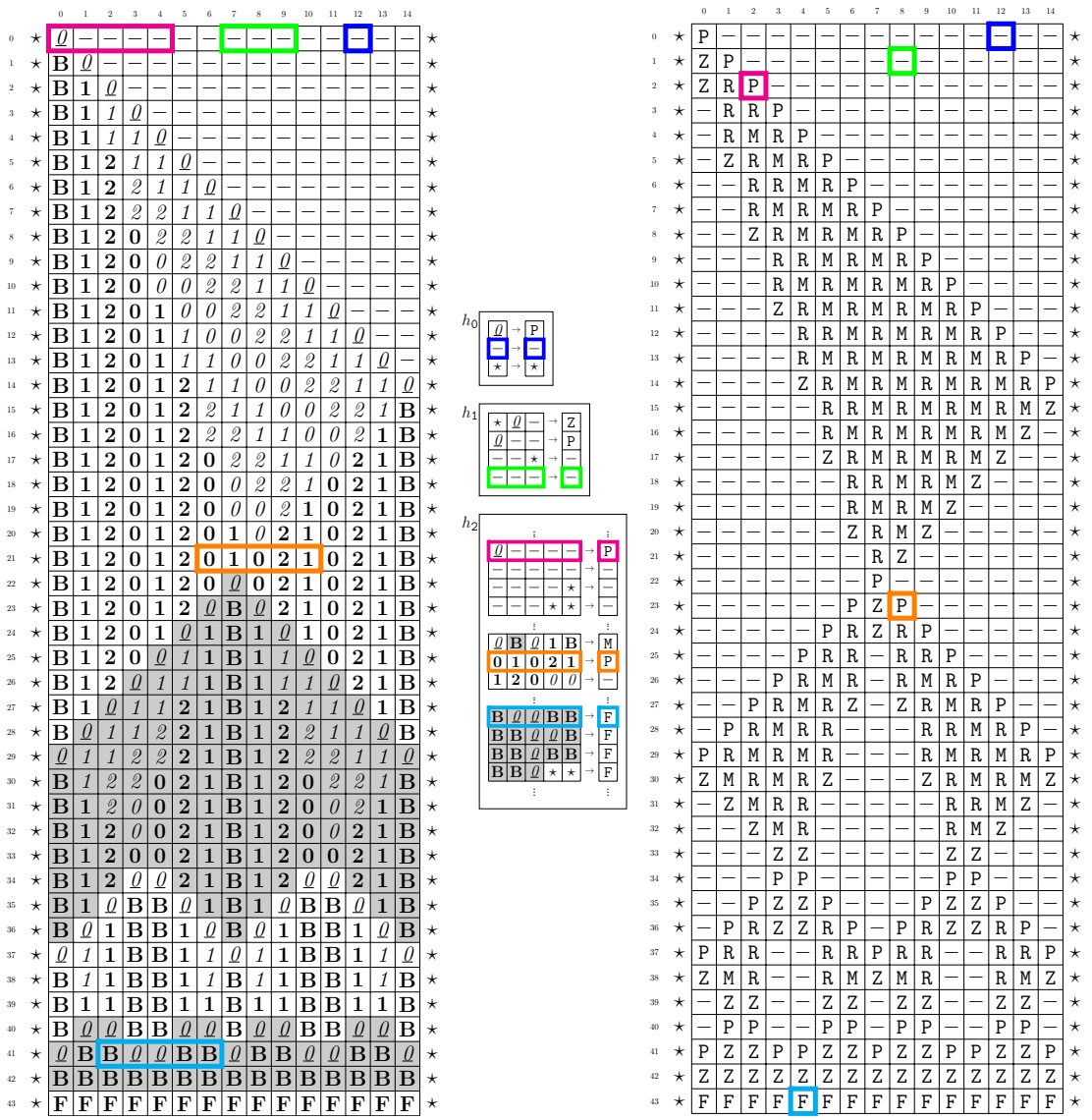
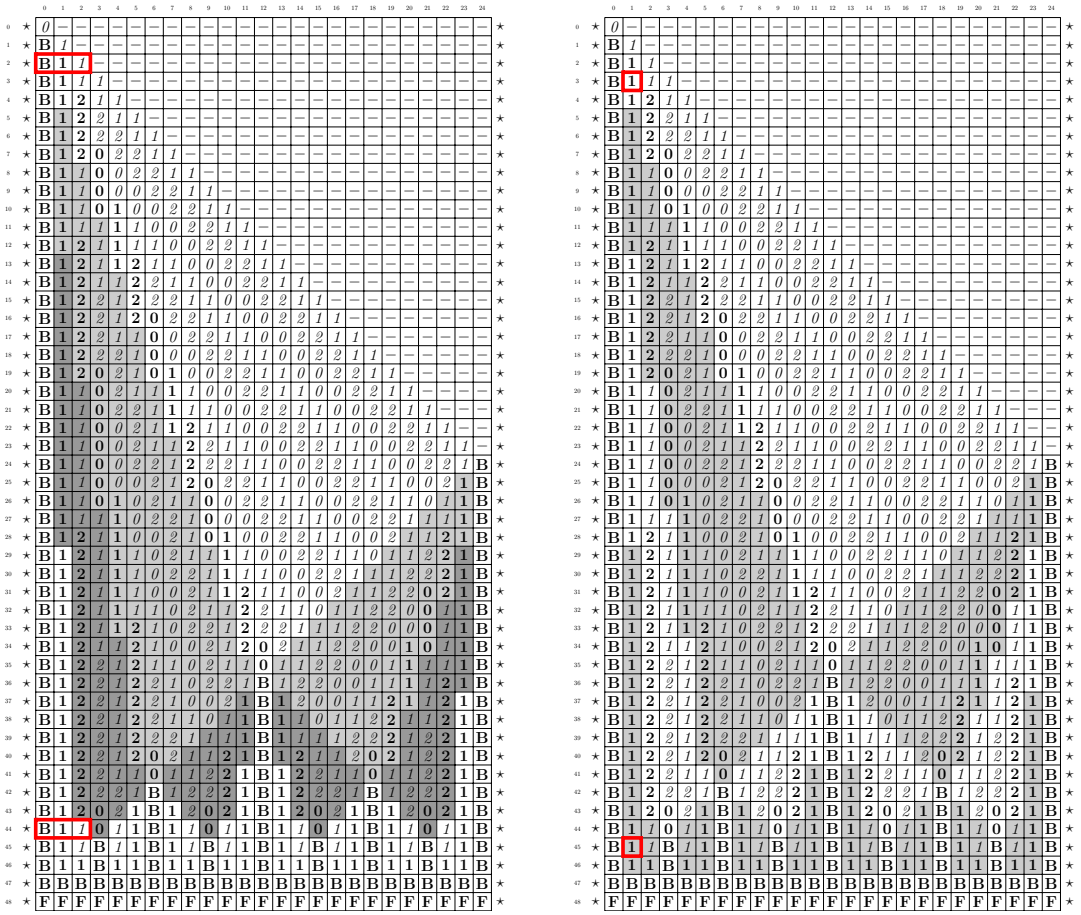


Figure 6.1: Local mapping $dt = 2$ from $\mathfrak{F}'^{18}_{373}$ to $\mathfrak{U}^{6}_{78}$ in FSSP-non-minimal-time.

For example, it is clear that using "mod 3" to obtain $\mathfrak{F}^{21}_{486}$ or "mod 2" to obtain $\mathfrak{F}^{15}_{329}$ means that these two solutions are very closely related. However, they are not local simulation in any direction, and for any depth a priori, because 2 and 3 are prime numbers. This is illustrated in Figure 6.2 for depth 1.

Of course, all of this is left as future work.



(a) $\mathfrak{F}^{21}_{486}$ (b) $\mathfrak{F}^{15}_{329}$

Figure 6.2: Pas de local mapping entre FB et FB2 de la taille 25 dans le cadre du FSSP (et non plus pour le sens inverse).

Bibliography

[1] Robert Balzer. An 8-state minimal time solution to the firing squad synchronization problem. *Information and Control*, 10(1):22–42, 1967.

[2] Tom Besson and Jérôme Durand-Lose. Exact discretization of 3-speed rational signal machines into cellular automata. In Matthew Cook and Turlough Neary, editors, *Cellular Automata and Discrete Complex Systems*, pages 63–76, Cham, 2016. Springer International Publishing.

[3] Manuel Clergue, Sébastien Vérel, and Enrico Formenti. An iterated local search to find many solutions of the 6-states firing squad synchronization problem. *Appl. Soft Comput.*, 66:449–461, 2018.

[4] Jean Duprat. Proof of correctness of the Mazoyer's solution of the firing squad problem in Coq. Research Report LIP RR-2002-14, Laboratoire de l'informatique du parallélisme, March 2002.

[5] Jérôme Durand-Lose. *Calculer géométriquement sur le plan - machines à signaux*. 2003.

[6] Patrick C. Fischer. Generation of primes by a one-dimensional real-time iterative array. *J. ACM*, 12(3):388–394, 1965.

[7] H. D. Gerken. Über Synchronisationsprobleme bei Zellularautomaten. *Diplomarbeit, Institut fur Theoretische Informatik, Technische Universität Braunschweig*, 50, 1987.

[8] GABOR T. HERMAN. Models for cellular interactions in development without polarity of individual cells ii. problems of synchronization and regulation. *International Journal of Systems Science*, 3(2):149–175, 1972.

[9] Joonatan Jalonen and Jarkko Kari. On the conjugacy problem of cellular automata. *Inf. Comput.*, 274:104531, 2020.

[10] Naoki Kamikawa and Hiroshi Umeo. A Study on Sequence Generation Powers of Small Cellular Automata. *SICE Journal of Control, Measurement, and System Integration*, 5(4):191–199, January 2012.

[11] Naoki Kamikawa and Hiroshi Umeo. A construction of five-state real-time fibonacci sequence generator. *Artif. Life Robotics*, 21(4):531–539, 2016.

[12] Naoki Kamikawa and Hiroshi Umeo. Two implementations of real-time sequence generator for {n^3 | n=1, 2, 3, ... } and their comparison. *Int. J. Netw. Comput.*, 9(2):257–275, 2019.

[13] Ivan Korec. Real-time generation of primes by a one-dimensional cellular automaton with 9-states. In Maurice Margenstern, editor, *International Colloquium Universal Machines and Computations, MCU'98, Metz, France, March 23-27, 1998, Proceedingsi, Volume II*, pages 101–116. IUT Metz, 1998.

[14] Luidnel Maignan. *Points, distances, et automates cellulaires : algorithmique géométrique et spatiale*. PhD thesis, 2010. Thèse de doctorat dirigée par Eisenbeis, Christine et Gruau, Frédéric, Informatique Paris 11 2010.

[15] Luidnel Maignan and Jean-Baptiste Yunès. A spatio-temporal algorithmic point of view on firing squad synchronisation problem. In Georgios Ch. Sirakoulis and Stefania Bandini, editors, *Cellular Automata - 10th International Conference on Cellular Automata for Research and Industry, ACRI 2012, Santorini Island, Greece, September 24-27, 2012. Proceedings*, volume 7495 of *Lecture Notes in Computer Science*, pages 101–110. Springer, 2012.

[16] Luidnel Maignan and Jean-Baptiste Yunès. Generalized fssp on hexagonal tiling: Towards arbitrary regular spaces. In Teijiro Isokawa, Katsunobu Imai, Nobuyuki Matsui, Ferdinand Peper, and Hiroshi Umeo, editors, *Cellular Automata and Discrete Complex Systems*, pages 83–96, Cham, 2015. Springer International Publishing.

[17] Luidnel Maignan and Jean-Baptiste Yunès. A field based solution of mazoyer's fssp schema. In Samira El Yacoubi, Jaroslaw Wąs, and Stefania Bandini, editors, *Cellular Automata*, pages 134–143, Cham, 2016. Springer International Publishing.

[18] Luidnel Maignan and Jean-Baptiste Yunès. Finitization of infinite field-based multi-general FSSP solution. *J. Cellular Automata*, 12(1-2):121–139, 2016.

[19] Luidnel Maignan and Jean-Baptiste Yunès. Moore and von neumann neighborhood n-dimensional generalized firing squad solutions using fields. In *2013 First International Symposium on Computing and Networking*, pages 552–558, 2013.

[20] Luidnel Maignan and Jean-Baptiste Yunès. Synchronizing the squad with (almost) arbitrary cut ratio. In *2016 Fourth International Symposium on Computing and Networking (CANDAR)*, pages 229–235, 2016.

[21] Jacques Mazoyer. A six-state minimal time solution to the firing squad synchronization problem. *Theor. Comput. Sci.*, 50:183–238, 1987.

[22] Marvin L. Minsky. Computation: finite and infinite machines. Prentice-Hall Series in Automatic Computation. Englewood Cliffs, NJ: Prentice-Hall, Inc. VII, 317 p. (1967)., 1967.

[23] Edward F. Moore. *Sequential Machines: Selected Papers*. Addison-Wesley Longman Ltd., GBR, 1964.

[24] Naoki Kamikawa and Hiroshi Umeo. Some algorithms for real-time generation of non-regular sequences on one-bit inter-cell-communication cellular automata. In *SICE Annual Conference 2007*, pages 953–958, 2007.

[25] Tien Thao Nguyen and Luidnel Maignan. Firsts steps in cellular fields optimization: A FSSP case study. In Giancarlo Mauri, Samira El Yacoubi, Alberto Dennunzio, Katsuhiro Nishinari, and Luca Manzoni, editors, *Cellular Automata - 13th International Conference on Cellular Automata for Research and Industry, ACRI 2018, Como, Italy, September 17-21, 2018, Proceedings*, volume 11115 of *Lecture Notes in Computer Science*, pages 264–273. Springer, 2018.

[26] Tien Thao Nguyen and Luidnel Maignan. Exploring millions of 6-state FSSP solutions: The formal notion of local CA simulation. In Hector Zenil, editor, *Cellular Automata and Discrete Complex Systems - 26th IFIP WG 1.5 International Workshop, AUTOMATA 2020, Stockholm, Sweden, August 10-12, 2020, Proceedings*, volume 12286 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2020.

[27] Tien Thao Nguyen and Luidnel Maignan. Some cellular fields interrelations and optimizations in FSSP solutions. *J. Cellular Automata*, 15(1-2):131–146, 2020.

[28] Tien Thao Nguyen and Luidnel Maignan. Millions of 5-state n^3 sequence generators via local mappings. *CoRR*, abs/2103.04626, 2021.

[29] Kenichiro Noguchi. Simple 8-state minimal time solution to the firing squad synchronization problem. *Theor. Comput. Sci.*, 314(3):303–334, 2004.

[30] Peter Sanders. Massively parallel search for transition-tables of polyautomata. In Chris R. Jesshope, Vesselin Jossifov, and Wolfgang Wilhelmi, editors, *Parcella 1994, VI. International Workshop on Parallel Processing by Cellular Automata and Arrays, Potsdam, Germany, September 21-23, 1994. Proceedings*, volume 81 of *Mathematical Research*, pages 99–108. Akademie Verlag, Berlin, 1994.

[31] Hiroshi Umeo. Recent developments in firing squad synchronization algorithms: Smaller solutions. In *Third International Conference on Networking and Computing, ICNC 2012, Okinawa, Japan, December 5-7, 2012*, pages 371–378. IEEE Computer Society, 2012.

[32] Hiroshi Umeo, Mitsuki Hirota, Youhei Nozaki, Keisuke Imai, and Takashi Sogabe. A new reconstruction and the first implementation of Goto's FSSP algorithm. *Appl. Math. Comput.*, 318:92–108, 2018.

[33] Hiroshi Umeo, Masaya Hisaoka, and Takashi Sogabe. A survey on optimum-time firing squad synchronization algorithms for one-dimensional cellular automata. *IJUC*, 1(4):403–426, 2005.

[34] Hiroshi Umeo, Masashi Maeda, and Kazuaki Hongyo. A design of symmetrical six-state $3n$-step firing squad synchronization algorithms and their implementations. In Samira El Yacoubi, Bastien Chopard, and Stefania Bandini, editors, *Cellular Automata, 7th International Conference on Cellular Automata, for Research and Industry, ACRI 2006, Perpignan, France, September 20-23, 2006, Proceedings*, volume 4173 of *Lecture Notes in Computer Science*, pages 157–168. Springer, 2006.

[35] Hiroshi Umeo, Masashi Maeda, Akihiro Sousa, and Kiyohisa Taguchi. A class of non-optimum-time 3n-step fssp algorithms - a survey. In Victor Malyshkin, editor, *Parallel Computing Technologies*, pages 231–245, Cham, 2015. Springer International Publishing.

[36] Hiroshi Umeo and Takashi Yanagihara. A smallest five-state solution to the firing squad synchronization problem. In Jérôme Olivier Durand-Lose and Maurice Margenstern, editors, *Machines, Computations, and Universality, 5th International Conference, MCU 2007, Orléans, France, September 10-13, 2007, Proceedings*, volume 4664 of *Lecture Notes in Computer Science*, pages 291–302. Springer, 2007.

[37] John von Neumann and Arthur W. Burks. *Theory of Self-Reproducing Automata*. University of Illinois Press, 1966.

[38] Abraham Waksman. An optimum solution to the firing squad synchronization problem. *Information and Control*, 9(1):66–78, 1966.

[39] Jean-Baptiste Yunès. Seven-state solutions to the firing squad synchronization problem. *Theor. Comput. Sci.*, 127(2):313–332, 1994.